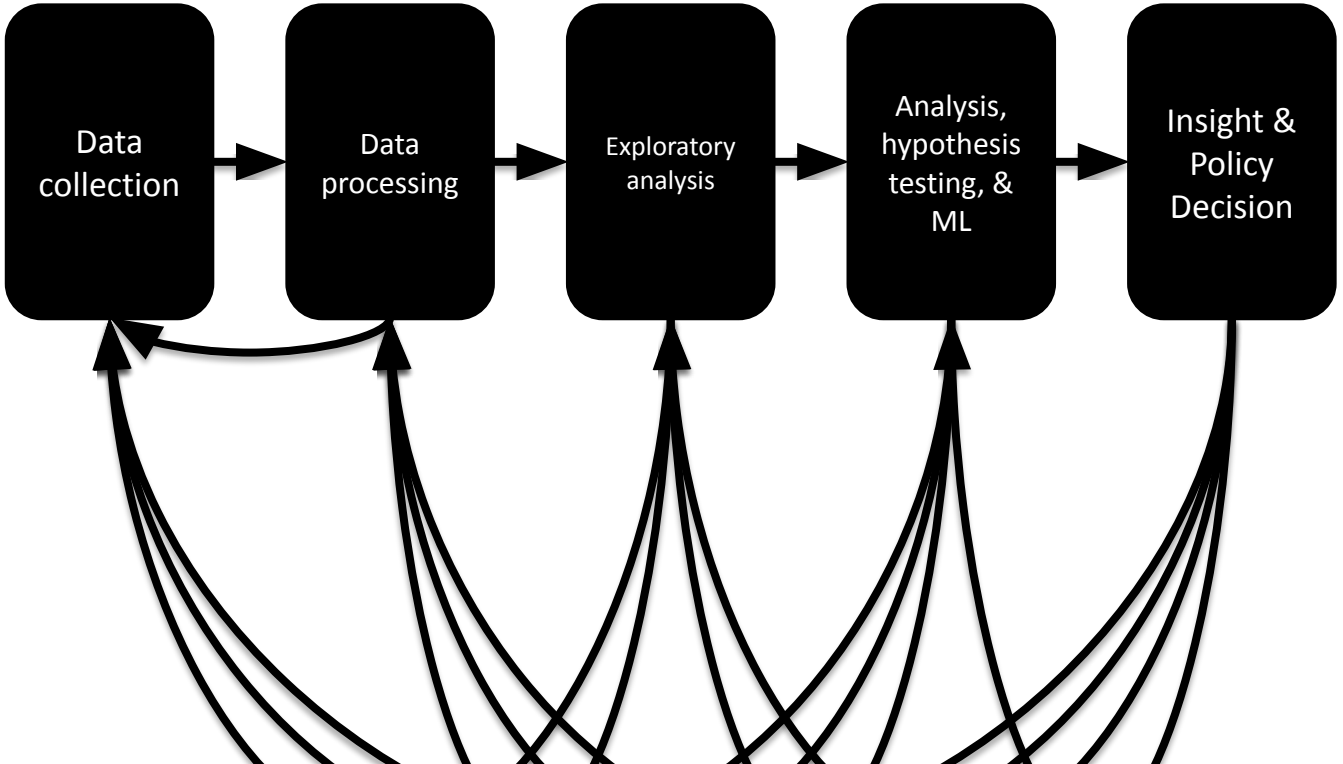
Data gathering, cleaning, transforming with Python

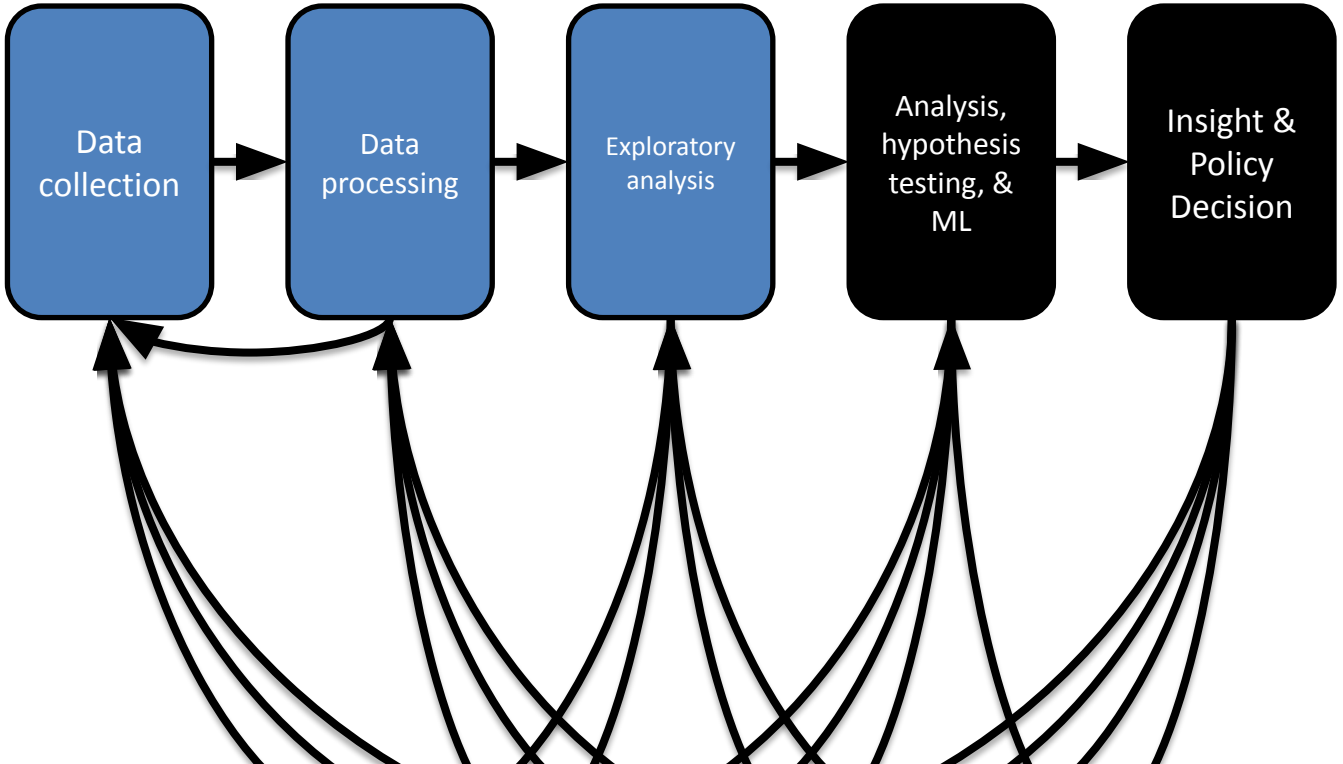
"The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that's going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids."

Hal Varian
Chief Economist at Google

The Data Lifecycle



The Data Lifecycle



How to get the data?

- Five ways to get data:
  - Direct download and load from local storage
  - Generate locally via downloaded code (e.g., simulation)
  - Query data from a database
  - Query an API from the intra/internet
  - Scrape data from a webpage

} Covered today

API

- A web-based Application Programming Interface (API) like we'll be using in this class is a contract between a server and a user stating:

"If you send me a specific request, I will return some information in a structured and documented format."

- (More generally, APIs can also perform actions, may not be web-based, be a set of protocols for communicating between processes, between an application and an OS, etc.)

“Send me a specific request”

- Most web API queries we’ll be doing will use HTTP requests:

```
r = requests.get( 'https://api.github.com/user',
                  auth=('user', 'pass')   )
```

```
r.status_code
```

```
200
```

```
r.headers['content-type']
```

```
'application/json; charset=utf8'
```

```
r.json()
```

```
{u'private_gists': 419, u'total_private_repos': 77, ...}
```

http://docs.python-requests.org/en/master/

HTTP Requests

- https://www.google.com/?q=cmsc320

What is this?

- **HTTP GET Request:**
- **GET /?q=cmsc320 HTTP/1.1**
  Host: www.google.com
  User-Agent: **Mozilla/5.0 (X11; Linux x86_64; rv:10.0.1) Gecko/20100101 Firefox/10.0.1**

```
params = { "q": "cmsc320" }
r = requests.get( "https://www.google.com",
                  params = params )
```

*be careful with https:// calls; `requests` will not verify SSL by default

"… I will return information in a structured format."

- So we've queried a server using a well-formed GET request via the `requests` Python module. What comes back?
- General structured data:
  - Comma-Separated Value (CSV) files & strings
  - Javascript Object Notation (JSON) files & strings
  - HTML, XHTML, XML files & strings
- Domain-specific structured data:
  - Shapefiles: geospatial vector data (OpenStreetMap)

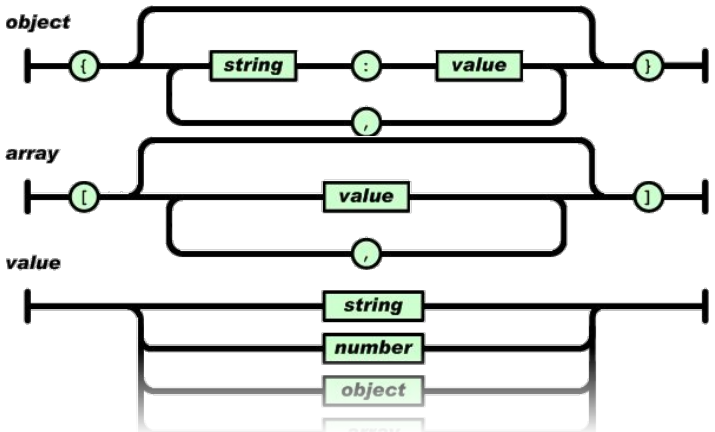
CSV Files in Python

- Any CSV reader worth anything can parse files with any delimiter, not just a comma (e.g., “TSV” for tab-separated)
- 1,26-Jan,Introduction,—,"pdf, pptx",Dickerson,
  2,31-Jan,Scraping Data with Python,Anaconda's Test Drive.,,Dickerson,
  3,2-Feb,"Vectors, Matrices, and Dataframes",Introduction to pandas.,,Dickerson,
  4,7-Feb,Jupyter notebook lab,,,"Denis, Anant, & Neil",
  5,9-Feb,Best Practices for Data Science Projects,,,Dickerson,
- Don’t write your own CSV parser

```
import csv
with open(“tips.csv”, ”rt”) as f:
     reader = csv.reader(f)
     for row in reader:
          print(row)
```

JSON Files & Strings

- JSON is a method for serializing objects:
  - Convert an object into a string
  - Deserialization converts a string back to an object
- Easy for humans to read (and sanity check, edit)
- Defined by three universal data structures



Python dictionary, Java Map, hash table, etc …

Python list, Java array, vector, etc …

Python string, float, int, boolean, JSON object, JSON array, …

Images from: http://www.json.org/

JSON In Python

- Some built-in types: `"Strings"`, `1.0`, `True`, `False`, `None`
- Lists: `["Goodbye", "Cruel", "World"]`
- Dictionaries: `{"hello": "bonjour", "goodbye": "au revoir"}`
- Dictionaries within lists within dictionaries within lists:
- ```
  [1, 2, {"Help":[
          "I'm", {"trapped": "in"},
          "CMSC320"
          ]}]
  ```
  ```

JSON From Twitter

- GET https://api.twitter.com/1.1/friends/list.json?cursor=-1&screen_name=twitterapi&skip_status=true&include_user_entities=false


```
{
        "previous_cursor": 0,
        "previous_cursor_str": "0",
        "next_cursor": 1333504313713126852,
        "users": [{
                "profile_sidebar_fill_color": "252429",
                "profile_sidebar_border_color": "181A1E",
                "profile_background_tile": false,
                "name": "Sylvain Carle",
                "profile_image_url":
"http://a0.twimg.com/profile_images/2838630046/4b82e286a659fae310012520f4f756bb_normal.png",
                "created_at": "Thu Jan 18 00:10:45 +0000 2007", …
```

Parsing JSON In Python

- Repeat: don't write your own JSON parser
  - https://news.ycombinator.com/item?id=7796268
  - rsdy.github.io/posts/dont_write_your_json_parser_plz.html
- Python comes with a fine JSON parser

```
import json
import requests

r = requests.get('https://api.github.com/events')
data = json.loads(r.text)
print(data[0]['repo']['url'])
print(data[0]['type'])
```

```
json.loads(some_file)  # loads JSON from a file
json.dump(json_obj, some_file)  # writes JSON to file
json.dumps(json_obj)  # returns JSON string
```

What is Web Scraping?

- A computer software technique of extracting information from websites
- Crawl sites and find information not easily accessible without robotic assistance
- Access data not contained in an API or a file

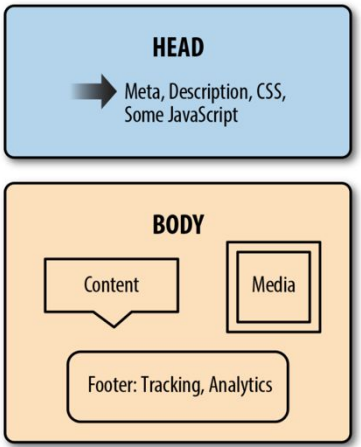
Web Scraping Outline

- What to Scrape and How
- Analyzing a Web Page
  - Inspection: Markup Structure
  - Network/Timeline: How the Page Loads
  - Console: Interacting with JavaScripts
  - IN-Depth Analysis of a Page
- Getting Pages: How to Request on the Internet
- Reading a Web Page with Beautiful Soup
- Reading a Web Page with LXML
  - A Case for XPath

웹 페이지가 어떤 구조로 이루어져있는지 파악
request 보내서 접근하는 방법 파악
웹페이지를 bs, LXML, XPath 등으로 긁어오기

What to Scrape and How

- What to scrape
  - Among millions of websites on the Internet with a huge variety of content and data
  - inform yourself about each site and what content you can scrape
- Understanding the structure of web pages is a prerequisite



Debugger on Chrome

Inspection: Markup Structure

- You can see series of nodes and their values which are HTML tags in Inspection tab

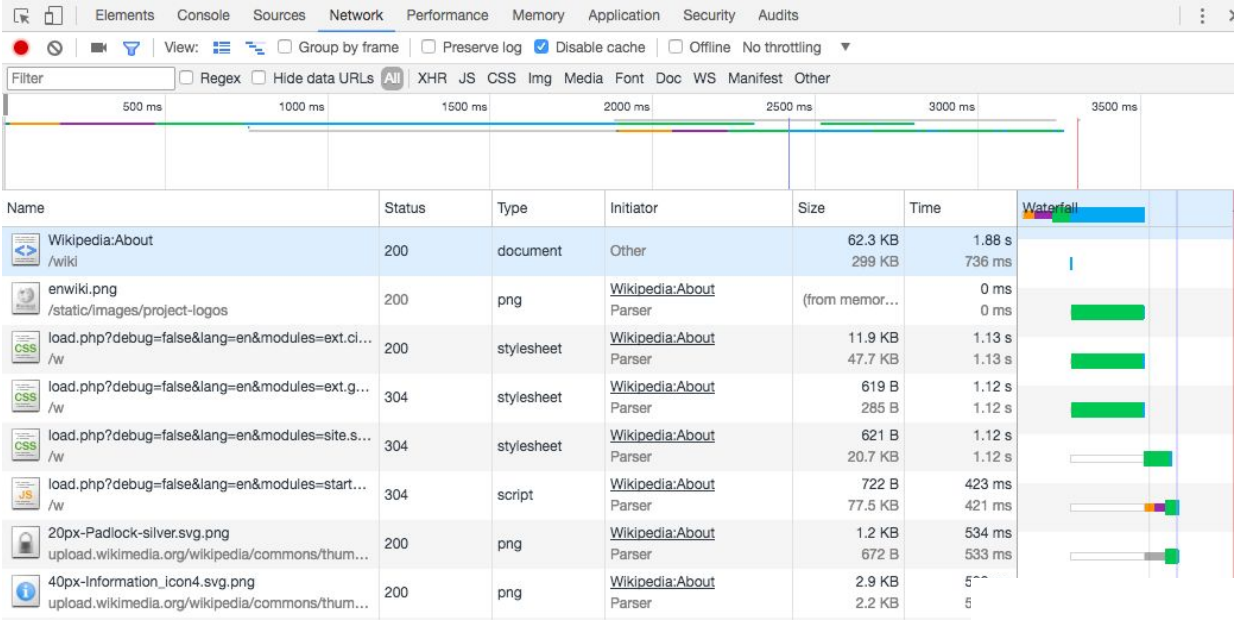
Tag	Description	Example
head	Used to hold metadata and other essential information for the document	<head> <title>Best Title Ever</ title> </head>
body	Used to hold the majority of the content on the page	<body> super short page </body>
meta	Used to hold metadata such as a short description of the site or keywords	<meta name="keywords" content="tags, html">
h1, h2, h3...	Used to hold header information; the smaller the number, the larger the header	Really big one!
p	Used to hold text paragraphs	Here's my first paragraph.
ul, ol	Used to hold both unordered lists (ul: think bullets) and ordered (ol: think numbers)	first bullet
li	Used to hold list items; should always be inside a list (ul or ol)	first second</ li>
div	Used to section or divide content	This div is about things.
a	Used to link content; called "anchor tags"	Best Ever
img	Used to insert an image	

<Basic HTML tags>

<Inspection tab Example>

Network/Timeline

- Show you how the contents on the page load, and in what order



Network/Timeline: How the Page Loads

- Most contents are loaded after the initial page load. □ Is there any call loading the content using JSON?
  - If so, you can find a URL with a JSON response holding the data you need, and use that URL to get the data
  - If not, or if the information is scattered over several different requests, you should use a browser-based approach to scrape the site

Console: Interacting with JavaScript

- Matching elements with JavaScript
  - getElementsByClassName(selector)
- Matching elements with jQuery
  - $(selector).action()

http://www.samsung.com/sec/

```
> document.getElementsByClassName('st-home');
◂ ▶ HTMLCollection [div#content.st-home, content: div#content.st-home]
> $('st-home');
◂ ▶ ie.fn.init [prevObject: ie.fn.init(1), context: document, selector: "st-home"]
```

Getting Pages: How to Request on the Internet

- First of all, you need to connect to the Web. urllib is Python's standard library for URL requests

```
import urllib
from urllib.request import urlopen

""" If you need to use a proxy server """
#######################################################
proxy_addr = "10.241.3.7:8080"
http_proxy = "http://" + proxy_addr
https_proxy = "https://" + proxy_addr

proxyDict = {"http": http_proxy, "https" : https_proxy}
proxy_support = urllib.request.ProxyHandler(proxyDict)
opener = urllib.request.build_opener(proxy_support)
urllib.request.install_opener(opener)
#######################################################

google = urllib.request.urlopen('https://google.com') #(1)
google = google.read() #(2)
print(google[:200]) #(3)

url = 'https://google.com/?q='
url_with_query = url + urllib.parse.quote_plus('python web scraping') #(4)

web_search = urllib.request.urlopen(url_with_query)
web_search = web_search.read()
print(web_search[:200])
```

1. urlopen() returns a buffer, where you can read the contents of the web page
2. Reads the contents of the entire page into the google variable
3. Prints the first 200 characters so we can see the beginning of the web page
4. quote_plus() transforms white space into '+'

Getting Pages: How to Request on the Internet

- requests library uses urllib and makes the complex requests easier to format and send

```
import requests

""" If you need to use a proxy server """
#########################################################
proxy_addr = "10.241.3.7:8080"
http_proxy = "http://" + proxy_addr
https_proxy = "https://" + proxy_addr

proxyDict = {"http": http_proxy, "https" : https_proxy}

r = requests.get('http://google.com', proxies=proxyDict) #(1)
#########################################################
# r = google.get('http://google.com')

print("status code = ", r.status_code)  #(2)
print("content = ", r.content[:200])
print("header = ", r.headers)  #(3)
print("cookie = ", r.cookies.items())  #(4)
```

1. Calls the get() method to send a GET request to the URL
2. Checks the status_code attribute to make sure we have 200 as response
3. Checks what headers Google sends back. We can see the headers attribute is a dictionary
4. Reads the cookies Google sends in a response

Reading a Web Page with Beautiful Soup

- Library for parsing HTML and XML documents
  - After fetching the data with requests or urllib, creates a parse tree for parsed pages

Parsing page source

```
import urllib
from urllib.request import urlopen
from bs4 import BeautifulSoup #(1)
from lxml import html

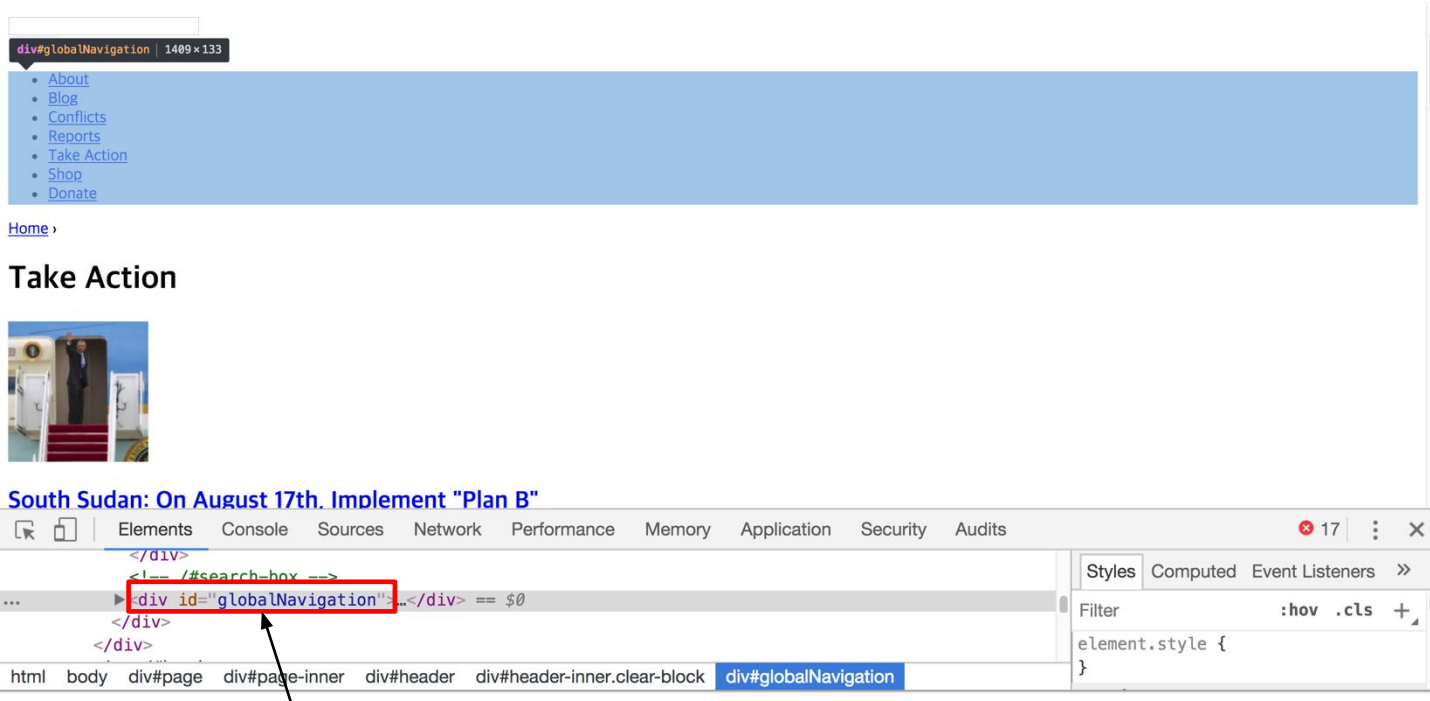
page = urlopen("file:///C:/path/to/take_action_enough_project.html") #(2)
bs = BeautifulSoup(page, "lxml") #(3)

print(bs.title)

print(bs.find_all('a')) #(4)

print(bs.find_all('p'))
```

1. Import the parser directly from the beautifulsoup4 library
2. Using urlopen() method from requests library to grab the content of the page. The variable *page* contains the content of the page
3. To start parsing with Beautiful Soup, we pass page source (content) of the *page* and lxml parser into the BeautifulSoup class
4. Now we can use its attributes and methods. Find all 'a' tags (or links) on the page



div element of id name “globalNavigation” contains menu bar of the page

Traversing relationships

- Uses descendants and siblings to traverse relationships

```
header_children = [c for c in bs.head.children] #(1)

print(header_children)

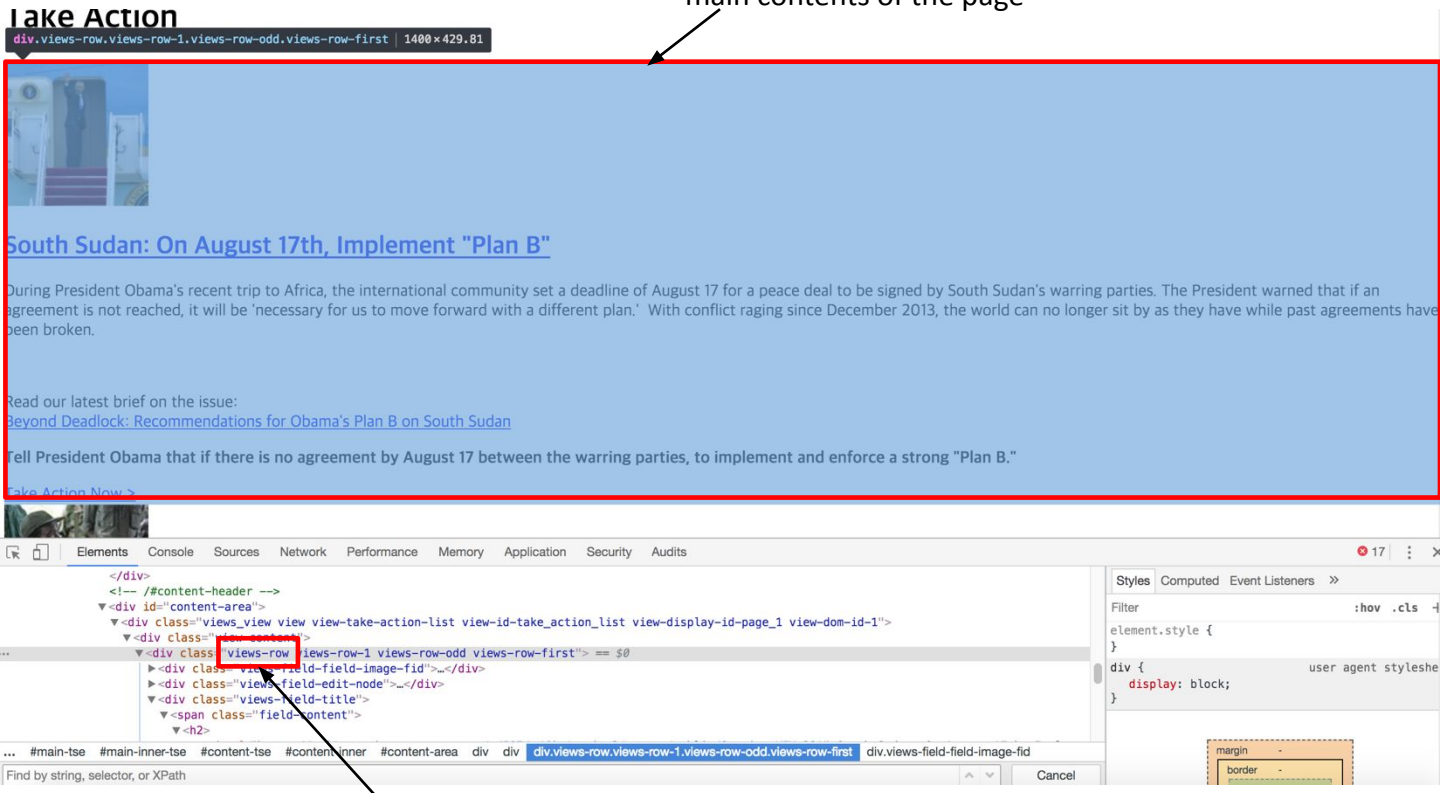
navigation_bar = bs.find(id="globalNavigation") #(2)

for d in navigation_bar.descendants: #(3)
    print(d)

for s in d.previous_siblings: #(4)
    print(s)
```

1. Create a list of all children from the header of the page
2. Passes CSS selector ID of global navigator to navigate family relationships
3. Iterate over the descendants of the navigation bar using the attribute descendants
4. Iterate over the siblings of the last descendant from our navigation bar

main contents of the page



div element of class name “views-row”
contains main contents of the page

Beautiful Soup Example

- Next, get title, link, and about information

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
from lxml import html

page = urlopen("file:///C:/path/to/take_action_enough_project.html")
bs = BeautifulSoup(page, "lxml")

ta_divs = bs.find_all("div", class_="views-row") #(1)
print("lengths of ta_divs = ", len(ta_divs)) #(2)

for ta in ta_divs:
    title = ta.h2 #(3)
    link = ta.a
    about = ta.find_all('p') #(4)
    print(title, link, about)
```

1. Find and return all divs tags whose class contains "views-row"
2. Prints the number of all divs tags
3. Iterates over the rows and grab the title and link
4. Find all paragraph tags

```
title =  <h2><a href="https://ssl1.americanprogress.org/o/507/p/dia/action3/common/public/?action_KEY=391">South Suda
n: On August 17th, Implement "Plan B" </a></h2>

link =  <a href="https://ssl1.americanprogress.org/o/507/p/dia/action3/common/public/?action_KEY=391">South Sudan: On
August 17th, Implement "Plan B" </a>

about =  [<p>During President Obama's recent trip to Africa, the international community set a deadline of August 17
 for a peace deal to be signed by South Sudan's warring parties. The President warned that if an agreement is not rea
ched, it will be 'necessary for us to move forward with a different plan.'  With conflict raging since December 2013,
the world can no longer sit by as they have while past agreements have been broken.</p>, <p> </p>, <p>Read our latest
brief on the issue:<br/>
<a href="http://eno.ug/1TtgaLd">Beyond Deadlock: Recommendations for Obama's Plan B on South Sudan</a></p>, <p><stron
g>Tell President Obama that if there is no agreement by August 17 between the warring parties, to implement and enfor
ce a strong "Plan B."</strong></p>]
```

- Instead of the entire element, we'd like to get the essential parts. Only texts of title, link, and paragraphs

```
all_data = []

for ta in ta_divs:
    data_dict = {}
    data_dict['title'] = ta.h2.get_text() #(1)
    data_dict['link'] = ta.a.get('href') #(2)
    data_dict['about'] = [p.get_text() for p in ta.find_all('p')] #(3)
    all_data.append(data_dict)

print(all_data)
```

Finally, we have a list of all the data within more organized format.

1. get_text() gives us all strings from the HTML element. So we can get a title text from the content of h2
2. To get an attribute of an element, we use get()
3. To extract the paragraph text, we use the get_text() method and iterate over the paragraphs returned by find_all()

```
title =  South Sudan: On August 17th, Implement "Plan B"

link =  https://ssl1.americanprogress.org/o/507/p/dia/action3/common/public/?action_KEY=391

about =  ["During President Obama's recent trip to Africa, the international community set a deadline of August 17 fo
r a peace deal to be signed by South Sudan's warring parties. The President warned that if an agreement is not reache
d, it will be 'necessary for us to move forward with a different\xa0plan.' \xa0With conflict raging since December 20
13, the world can no longer sit by as they have while past agreements have been broken.", '\xa0', "Read our latest br
ief on the issue:\nBeyond Deadlock: Recommendations for Obama's Plan B on South Sudan", 'Tell President Obama that if
 there is no agreement by\xa0August 17 between the warring parties, to implement and enforce a strong "Plan\xa0B."']
```

Reading a Web Page with LXML

- Python library for processing XML and HTML
  - It can traverse the DOM(Document Object Model) and family relationships using css selectors
  - It has a lot of great features, including ability to generate HTML and XML documents to clean up poorly written pages

Document Object Model:
- Programming model for HTML and XML documents.
DOM provides structured representation of a document and allows us to access a DOM structure with programming languages

LXML Example

- Rewriting the Beautiful Soup code to use lxml

```
from lxml import html

page = html.parse("file:///C:/path/to/html/take_action_enough_project.html") #(1)
root = page.getroot() #(2)

ta_divs = root.cssselect('div.views-row') #(3)
print("ta_divs = ", ta_divs)

all_data = []

for ta in ta_divs:
    data_dict = {}
    title = ta.cssselect('h2')[0] #(4)
    data_dict['title'] = title.text_content() #(5)
    data_dict['link'] = title.find('a').get('href') #(6)
    data_dict['about'] = [p.text_content() for p in ta.cssselect('p')] #(7)
    all_data.append(data_dict)
print("all_data = ", all_data)
```

Use local HTML file instead of a URL

1. Using lxml's parsing method
2. Access the top of the page and HTML, root. The root contains all of the possible branches(children) and twigs(descendants) within reach
3. Find all of the divs with class 'views-row'. The cssselect() method returns a list of elements which are matched with given CSS selector string
4. To grab the titles, we use cssselect('h2')
5. text_content() returns text
6. To grab the link, we use attribute name 'href' and pull that attribute from the anchor tag
7. Similar to Beautiful Soup, we iterate over the paragraphs returned from cssselect()

find() vs. cssselect()

- Main difference between find() and cssselect()
  - find() traverses the elements of the DOM using ancestry and familial relationships
  - cssselect() utilizes CSS selectors to find all possible matches within the page or the elements' descendants

```
print(root.find('div')) #(1)

print(root.find('head'))

print(root.find('head').findall('script')) #(2)

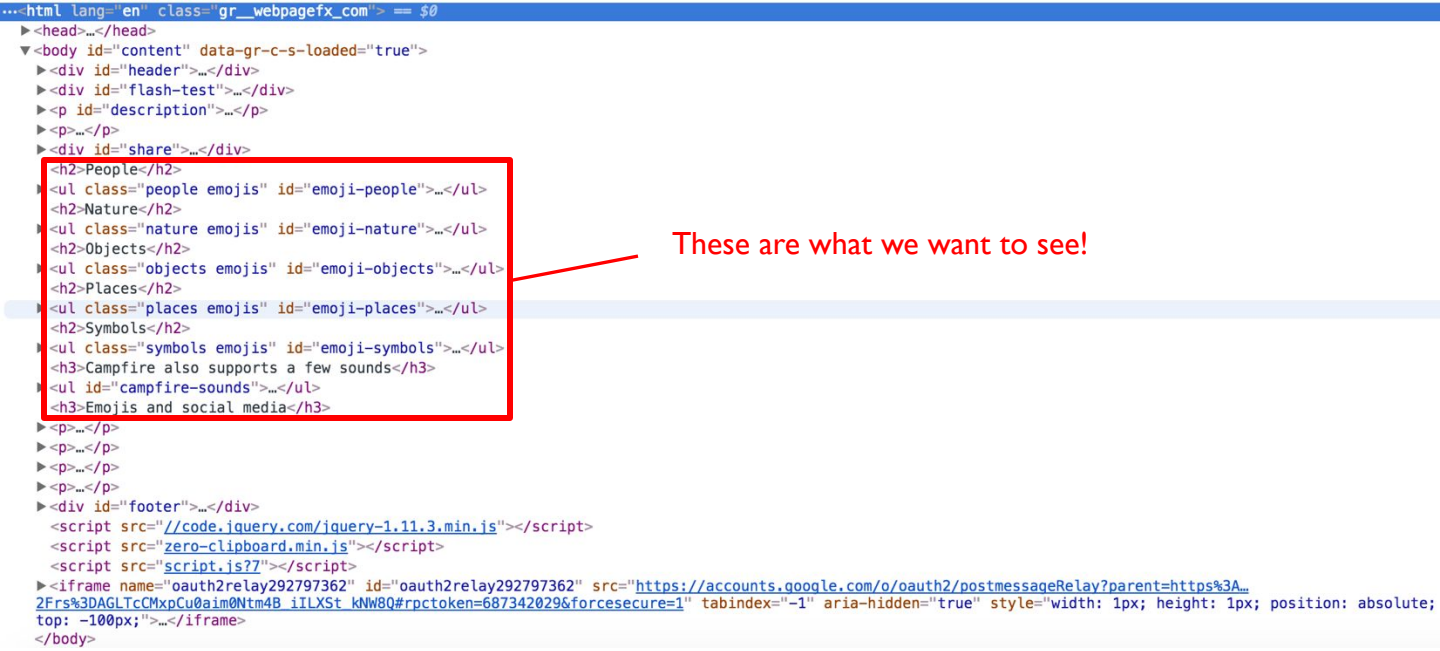
print(root.cssselect('div')) #(3)

print(root.cssselect('head script')) #(4)
```

1. find('div') returns *child* elements of the root whose types are 'div'
2. findall('script') returns descendant elements whose types are 'script'
3. Uses cssselect() to locate all 'divs' contained in the document
4. Uses cssselect() to locate the script tags within the header section

LXML Example

- Let’s see an example of HTML document of the page ‘http://www.emoji-cheat-sheet.com/’



- Design a parser which returns a list of emojis

```
from lxml import html
import requests

page = html.parse('file:///C:/path/to/html/emoji_cheat_sheet.html')
root = page.getroot()

body = root.find('body') #(1)
top_header = body.find('h2')

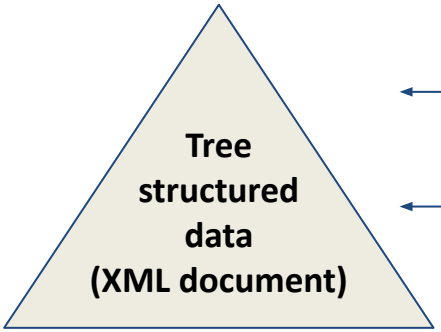
print("top_header.text = ", top_header.text)

headers_and_lists = [sib for sib in top_header.itersiblings()] #(2)
print("\nheaders_and_lists = ", headers_and_lists)
proper_headers_and_lists = [s for s in top_header.itersiblings()
                            if s.tag in ['ul', 'h2', 'h3']] #(3)

print("\nproper_headers_and_lists = ", proper_headers_and_lists)
```

Tree navigation

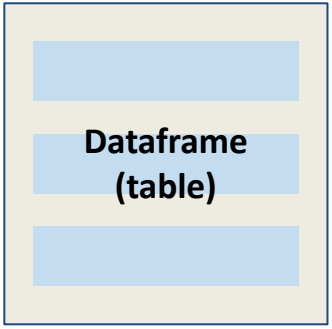
1. By viewing the page structure, we know series of headers to see. Uses the find() method to locate the first header in order to use familial relationships to find the other headers
2. The method itersiblings() allows us to see all of the siblings
3. We can choose what we need. In this case, we determine that the only tags we want are 'ul', 'h2' and 'h3'



API

Language
(XPath, XQuery)

XQuery > SQL > XPath 순으로 파워가 쎔



API

Language
(SQL)

Contains needless elements like this

```
headers_and_lists =  [<Element ul at 0x7fe1b32154a8>, <Element h2 at 0x7fe1b3215408>, <Element ul at 0x7fe1b3215818>,
<Element h2 at 0x7fe1b3215868>, <Element ul at 0x7fe1b32159a8>, <Element h2 at 0x7fe1b3215a48>, <Element ul at 0x7fe1
b3215a98>, <Element h2 at 0x7fe1b322fae8>, <Element ul at 0x7fe1b322fb88>, <Element h3 at 0x7fe1b322fbd8>, <Element u
l at 0x7fe1b305b1d8>, <Element h3 at 0x7fe1b305b3b8>, <Element p at 0x7fe1b306f0e8>, <Element p at 0x7fe1b306f138>, <
Element p at 0x7fe1b306f318>, <Element p at 0x7fe1b306f6d8>, <Element div at 0x7fe1b306f408>, <Element script at 0x7f
e1b306f688>, <Element script at 0x7fe1b306f638>, <Element script at 0x7fe1b306f778>]

proper_headers_and_lists =  [<Element ul at 0x7fe1b32154a8>, <Element h2 at 0x7fe1b3215408>, <Element ul at 0x7fe1b32
15818>, <Element h2 at 0x7fe1b3215868>, <Element ul at 0x7fe1b32159a8>, <Element h2 at 0x7fe1b3215a48>, <Element ul a
t 0x7fe1b3215a98>, <Element h2 at 0x7fe1b322fae8>, <Element ul at 0x7fe1b322fb88>, <Element h3 at 0x7fe1b322fbd8>, <E
lement ul at 0x7fe1b305b1d8>, <Element h3 at 0x7fe1b305b3b8>]
```

Remove needless elements

What is XPath?

- A standard query language for XML documents

Principal Kinds of Nodes

1. *Document nodes* represent entire documents.
2. *Elements* are pieces of a document consisting of some opening tag, its matching closing tag (if any), and everything in between.
3. *Attributes* names that are given values inside opening tags.

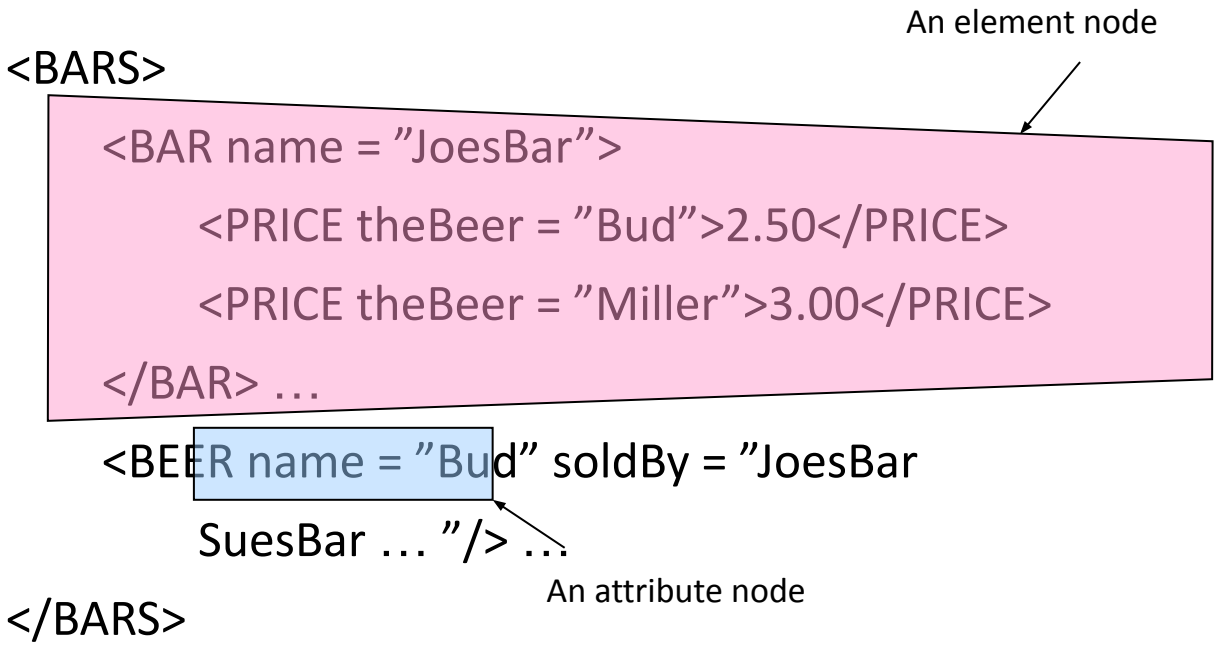
Document Nodes

- Formed by doc(URL) or document(URL).
- Example: doc(/usr/class/cs145/bars.xml)
- All XPath (and XQuery) queries refer to a doc node, either explicitly or implicitly.
  - Example: key definitions in XML Schema have Xpath expressions that refer to the document described by the schema.

DTD for Running Example

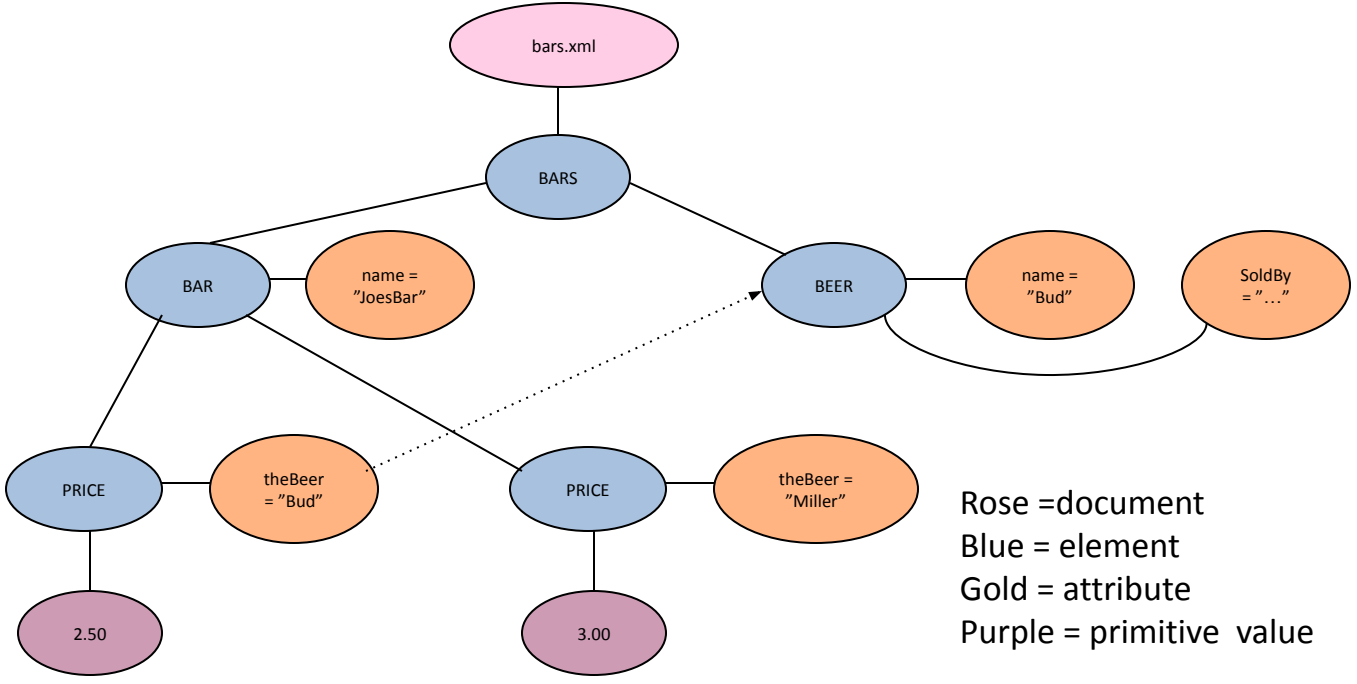
```
<!DOCTYPE BARS [
        <!ELEMENT BARS (BAR*, BEER*)>
        <!ELEMENT BAR (PRICE+)>
                <!ATTLIST BAR name ID #REQUIRED>
        <!ELEMENT PRICE (#PCDATA)>
                <!ATTLIST PRICE theBeer IDREF #REQUIRED>
        <!ELEMENT BEER EMPTY>
                <!ATTLIST BEER name ID #REQUIRED>
                <!ATTLIST BEER soldBy IDREFS #IMPLIED>
]>
```

Example Document



Document node is all of this, plus the header (<? xml version…).

Nodes as Semistructured Data



Rose =document
Blue = element
Gold = attribute
Purple = primitive value

Paths in XML Documents

- XPath is a language for describing paths in XML documents.
- The result of the described path is a sequence of items.

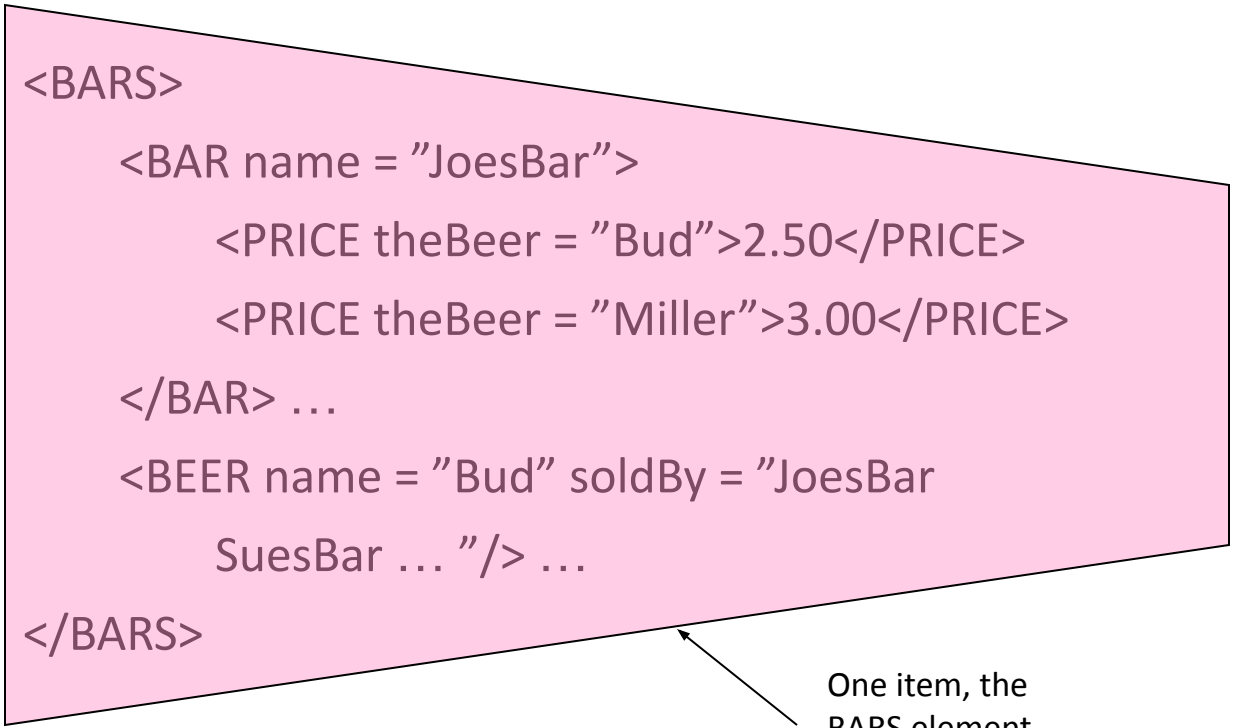
Path Expressions

- Simple path expressions are sequences of slashes (/) and tags, starting with /.
  - Example: /BARS/BAR/PRICE
- Construct the result by starting with just the doc node and processing each tag from the left.

Evaluating a Path Expression

- Assume the first tag is the root.
  - Processing the doc node by this tag results in a sequence consisting of only the root element.
- Suppose we have a sequence of items, and the next tag is *X*.
  - For each item that is an element node, replace the element by the subelements with tag *X*.

Example: /BARS



Example: /BARS/BAR

<BARS>

<BAR name = ”JoesBar”>

<PRICE theBeer =”Bud”>2.50</PRICE>

<PRICE theBeer = ”Miller”>3.00</PRICE>

</BAR> …

<BEER name = ”Bud” soldBy = ”JoesBar SuesBar …”/> …

</BARS>

This BAR element followed by all the other BAR elements

Example: /BARS/BAR/PRICE

<BARS>

<BAR name = ”JoesBar”>

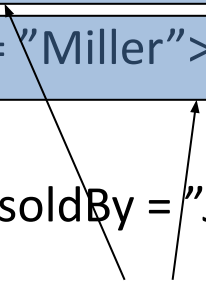
<PRICE theBeer =”Bud”>2.50</PRICE>

<PRICE theBeer = ”Miller”>3.00</PRICE>

</BAR> …

<BEER name = ”Bud” soldBy = ”JoesBar SuesBar …”/> …

</BARS>



These PRICE elements followed by the PRICE elements of all the other bars.

Attributes in Paths

- Instead of going to subelements with a given tag, you can go to an attribute of the elements you already have.
- An attribute is indicated by putting @ in front of its name.

Example: /BARS/BAR/PRICE/@theBeer

<BARS>

<BAR name = ”JoesBar”>

<PRICE theBeer = ”Bud”>2.50</PRICE>

<PRICE theBeer = ”Miller”>3.00</PRICE>

</BAR> …

<BEER name = ”Bud” soldBy = ”JoesBar

SuesBar …”/>

</BARS>

These attributes contribute ”Bud” ”Miller” to the result, followed by other theBeer values.

Remember: Item Sequences

- Until now, all item sequences have been sequences of elements.
- When a path expression ends in an attribute, the result is typically a sequence of values of primitive type, such as strings in the previous example.

Paths that Begin Anywhere

- If the path starts from the document node and begins with //*X*, then the first step can begin at the root or any subelement of the root, as long as the tag is *X*.

Example: //PRICE

<BARS>

<BAR name = ”JoesBar”>

<PRICE theBeer =”Bud”>2.50</PRICE>

<PRICE theBeer = ”Miller”>3.00</PRICE>

</BAR> …

<BEER name = ”Bud” soldBy = ”JoesBar SuesBar …”/> …

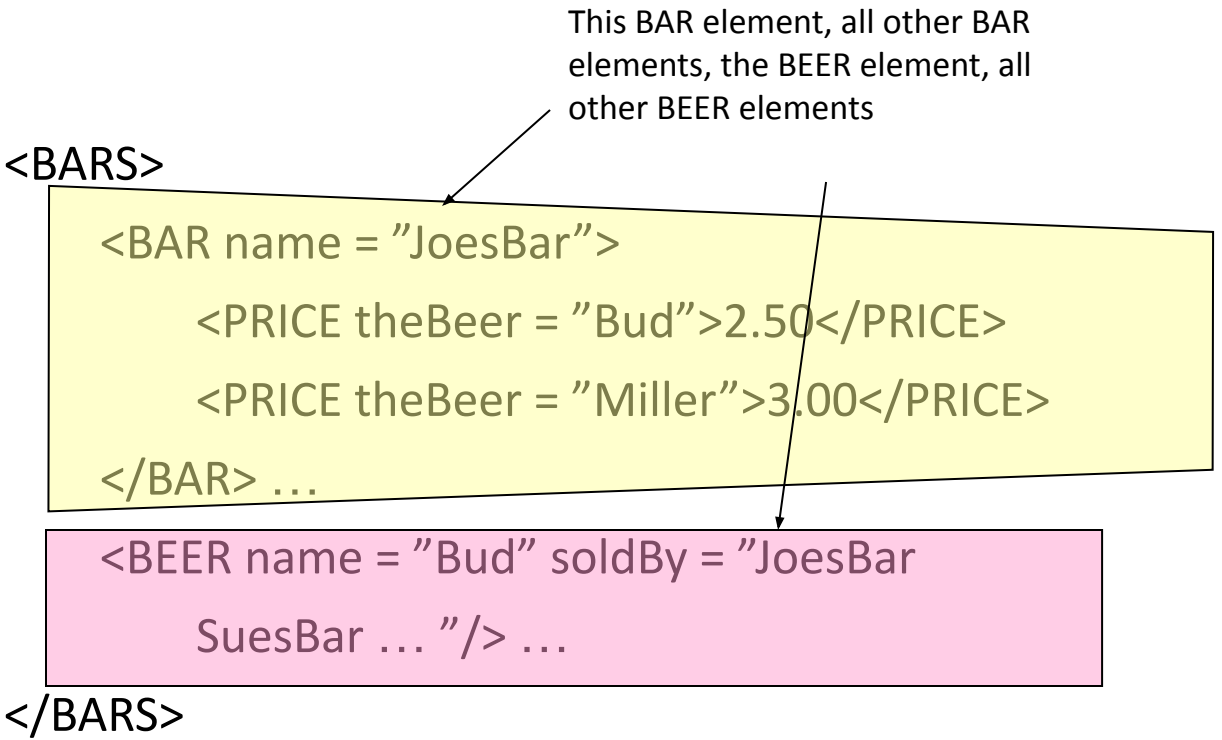
</BARS>

These PRICE elements and any other PRICE elements in the entire document

Wild-Card *

- A star (*) in place of a tag represents any one tag.
- Example: /*/*/PRICE represents all price objects at the third level of nesting.

Example: /BARS/*



Selection Conditions

- A condition inside […] may follow a tag.
- If so, then only paths that have that tag and also satisfy the condition are included in the result of a path expression.

Example: Selection Condition

- /BARS/BAR/PRICE[. < 2.75]

<BARS>

<BAR name = ”JoesBar”>

<PRICE theBeer = ”Bud”>2.50</PRICE>

<PRICE theBeer = ”Miller”>3.00</PRICE>

</BAR> …

The condition that the PRICE be < $2.75 makes this price but not the Miller price part of the result.

Example: Attribute in Selection

- /BARS/BAR/PRICE[@theBeer = ”Miller”]

<BARS>

<BAR name = ”JoesBar”>

<PRICE theBeer = ”Bud”>2.50</PRICE>

<PRICE theBeer = ”Miller”>3.00</PRICE>

</BAR> …

Now, this PRICE element
is selected, along with
any other prices for Miller.

- /BARS/BAR/PRICE[@theBeer = ”Miller”].text()

3.00

Axes

- In general, path expressions allow us to start at the root and execute steps to find a sequence of nodes at each step.
- At each step, we may follow any one of several *axes*.
- The default axis is child:: --- go to all the children of the current set of nodes.

Example: Axes

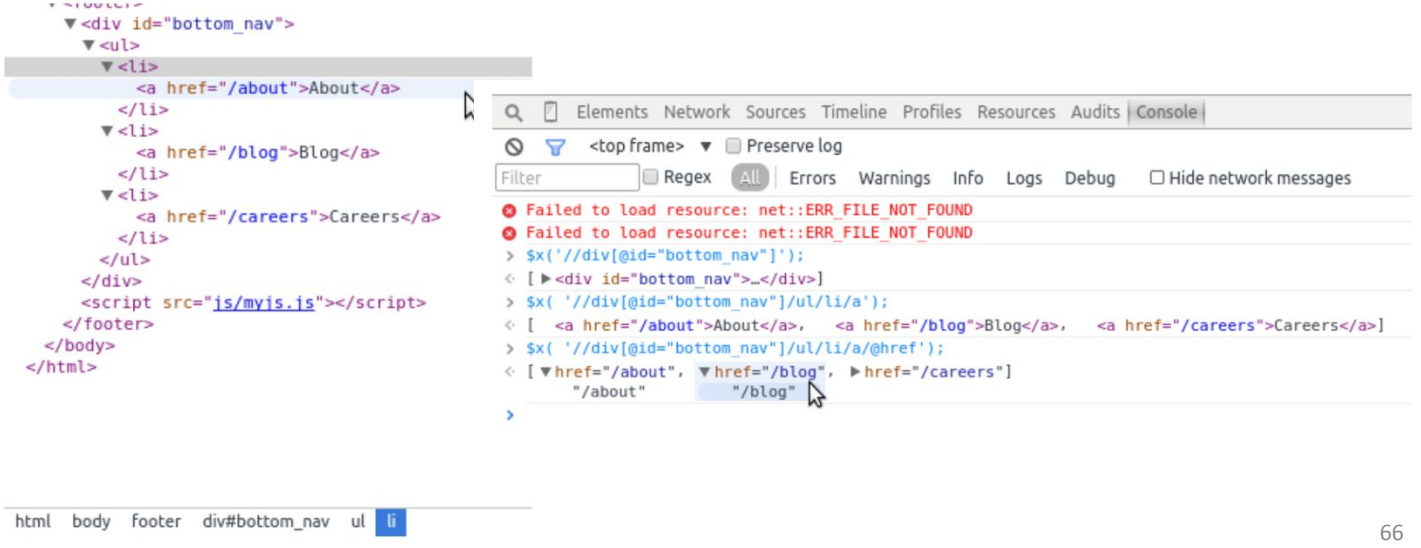
- /BARS/BEER is really shorthand for /BARS/child::BEER .
- @ is really shorthand for the attribute:: axis.
  - Thus, /BARS/BEER[@name = ”Bud”] is shorthand for /BARS/BEER[attribute::name = ”Bud”]

More Axes

- Some other useful axes are:
  1. parent:: = parent(s) of the current node(s).
  2. descendant-or-self:: = the current node(s) and all descendants.
     - Note: // is really shorthand for this axis.
  3. ancestor::, ancestor-or-self, etc.
  4. self (the dot).

XPath Syntax Example

- ‘//div[@id=“bottom_nav”]/ul/li/a/@href’



XPath Example

- get all headers and list of data

```
from lxml import html

page = html.parse('file:///C:/path/to/html/emoji_cheat_sheet.html')

proper_headers = page.xpath('//h2|//h3') #(1)
proper_lists = page.xpath('//ul') #(2)

print("proper_headers = ", proper_headers)
print("\nproper_lists = ", proper_lists)
```

1. It returns all 'h2' and 'h3' elements which are headers related to emoji content.
2. It returns all 'ul' elements in the entire document.

```
proper_headers =  [<Element h2 at 0x7fe1b3215908>, <Element h2 at 0x7fe1b32281d8>, <Element h2 at 0x7fe1b3228278>, <E
lement h2 at 0x7fe1b3228228>, <Element h2 at 0x7fe1b3228188>, <Element h3 at 0x7fe1b3228098>]

proper_lists =  [<Element ul at 0x7fe1b32282c8>, <Element ul at 0x7fe1b3228318>, <Element ul at 0x7fe1b3228368>, <Ele
ment ul at 0x7fe1b32283b8>, <Element ul at 0x7fe1b3228408>, <Element ul at 0x7fe1b3228458>]
```

Advanced Web Scraping Outline

- Browser-Based Scraping
  - Screen Reading with Selenium
- Spidering (crawling) the Web
  - Building a Spider with Scrapy
  - Crawling Whole Websites with Scrapy

Step for the Web Scraping

- Many web pages offer contents on their first page load
  - For these pages, a simple XML or HTML parser is sufficient
- If you need to interact with the page to get the data dynamically
  - You need a Browser-based scraper
- If you need to traverse an entire web site
  - You need a robot that crawls pages called spider

Browser-Based Scraping

- Normal web scraper cannot analyze a site
  - If the site uses a lot of JavaScript
  - If the site uses post-page-load codes
  - If you want to interact with pages (i.e., click the button or enter some search text)
- In this case, you need to use Screen readers which are used for ‘opening’, ‘reading’ and ‘interacting’ with web pages

Screen Reading with Selenium

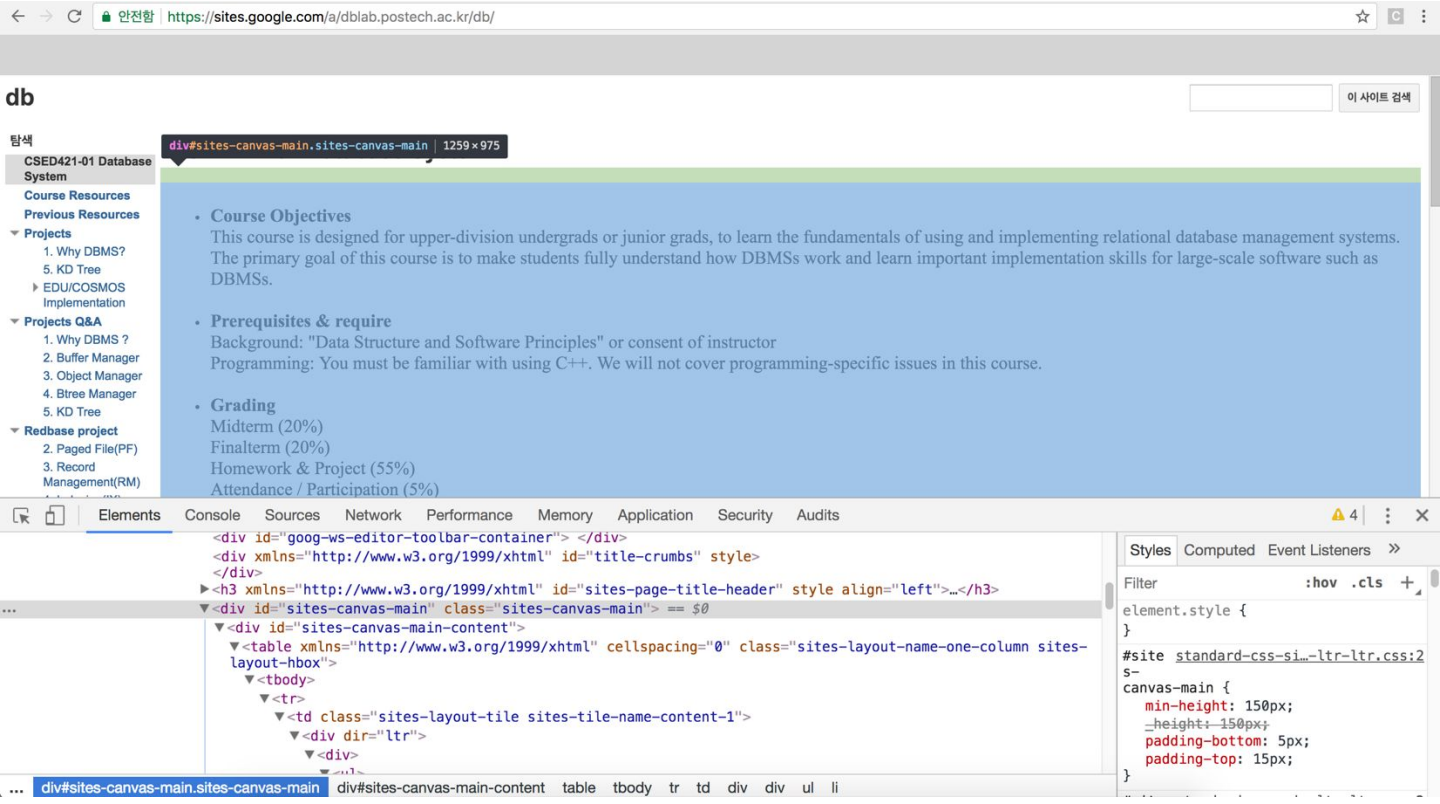
- What is Selenium?
  - Powerful Java-based engine to interact directly with a website through the Selenium-supported browsers
  - Very popular framework for user testing

Opening browser

```
from selenium import webdriver

browser = webdriver.Chrome()
browser.get('http://sites.google.com/a/dblab.postech.ac.kr/db/')
```

1. Imports the webdriver module from Selenium. This module is used to call any installed drivers
2. Open a new Chrome window on your computer
3. Accesses the URL with get()


안전함 https://sites.google.com/a/dblab.postech.ac.kr/db/
db
이 사이트 검색
탐색
CSED421-01 Database System
Course Resources
Previous Resources
Projects
1. Why DBMS?
5. KD Tree
EDU/COSMOS Implementation
Projects Q&A
1. Why DBMS ?
2. Buffer Manager
3. Object Manager
4. Btree Manager
5. KD Tree
Redbase project
2. Paged File(PF)
3. Record Management(RM)
div#sites-canvas-main.sites-canvas-main 1259 × 975
Course Objectives
This course is designed for upper-division undergrads or junior grads, to learn the fundamentals of using and implementing relational database management systems. The primary goal of this course is to make students fully understand how DBMSs work and learn important implementation skills for large-scale software such as DBMSs.
Prerequisites & require
Background: "Data Structure and Software Principles" or consent of instructor
Programming: You must be familiar with using C++. We will not cover programming-specific issues in this course.
Grading
Midterm (20%)
Finalterm (20%)
Homework & Project (55%)
Attendance / Participation (5%)
Elements Console Sources Network Performance Memory Application Security Audits

<div xmlns="http://www.w3.org/1999/xhtml" id="title-crumbs" style>

<h3 xmlns="http://www.w3.org/1999/xhtml" id="sites-page-title-header" style align="left">…</h3>
 == $0

<table xmlns="http://www.w3.org/1999/xhtml" cellspacing="0" class="sites-layout-name-one-column sites-layout-hbox">
<tbody>
<tr>
<td class="sites-layout-tile sites-tile-name-content-1">

div#sites-canvas-main.sites-canvas-main div#sites-canvas-main-content table tbody tr td div div ul li
Styles Computed Event Listeners
Filter :hov .cls
element.style {
}
#site standard-css-si…-ltr-ltr.css:2
s-
canvas-main {
min-height: 150px;
height: 150px;
padding-bottom: 5px;
padding-top: 15px;
}

Interacting with page

- Finds content of the page using CSS selector

```
>>> content = browser.find_element_by_css_selector('div.sites-canvas-main')
>>> print(content.text)
Course Objectives
This course is designed for upper-division undergrads or junior grads, to learn t
he fundamentals of using and implementing relational database management systems.
 The primary goal of this course is to make students fully understand how DBMSs w
ork and learn important implementation skills for large-scale software such as DB
MSs.

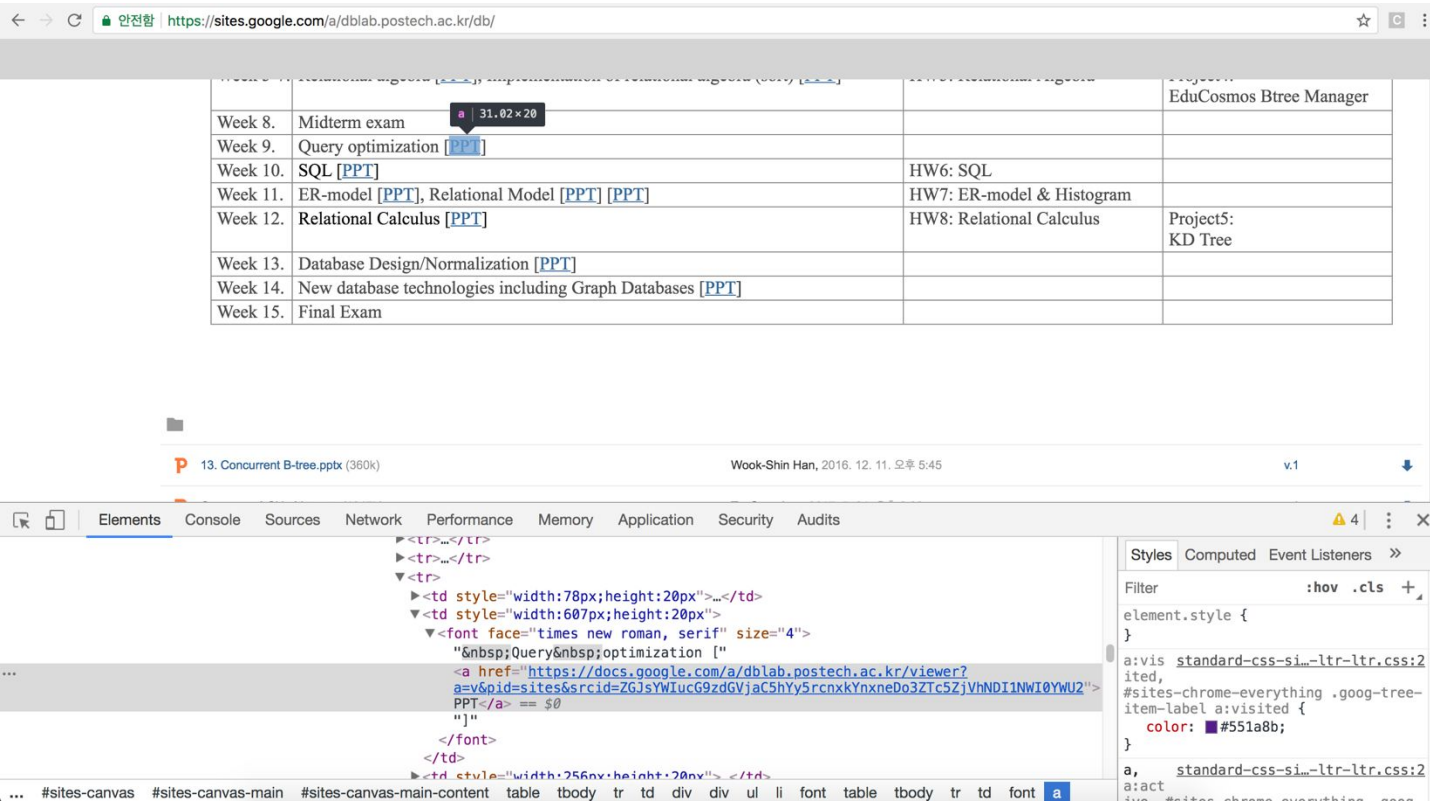
Prerequisites & require
Background: "Data Structure and Software Principles" or consent of instructor
Programming: You must be familiar with using C++. We will not cover programming-s
pecific issues in this course.

Grading
Midterm (20%)
Finalterm (20%)
Homework & Project (55%)
Attendance / Participation (5%)
Course Materials
Ramakrishnan and Gehrke, Database Management Systems, 3rd ed.

Course Plan
 Week  Topic   Homework  Projects
 Week 1.  Introduction [PPT]  HW1: Data Independence  Project 1:
 Why DBMS?
 Week 2.  Disk and Files [PPT]    Project 2:
 EduCosmos Buffer Manager
 Week 3.  Indexing [PPT] [PPT2], Concurrent B-tree [PPT]
 HW2: B+ tree
```

⋮

1. The browser object calls find_element_by_css_selector() that uses CSS selectors to select HTML elements. It returns the first match
2. Print the text of the first matched element



			EduCosmos Btree Manager
Week 8.	Midterm exam		
Week 9.	Query optimization [PPT]		
Week 10.	SQL [PPT]	HW6: SQL	
Week 11.	ER-model [PPT], Relational Model [PPT] [PPT]	HW7: ER-model & Histogram	
Week 12.	Relational Calculus [PPT]	HW8: Relational Calculus	Project5: KD Tree
Week 13.	Database Design/Normalization [PPT]		
Week 14.	New database technologies including Graph Databases [PPT]		
Week 15.	Final Exam		

Selenium with XPath

- Lists all links in the page using xpath

```
>>> links = browser.find_elements_by_xpath('//a')
>>> for link in links:
...     print(link.get_attribute("href"))
...
https://sites.google.com/a/dblab.postech.ac.kr/db/
https://sites.google.com/a/dblab.postech.ac.kr/db/course-resources
https://sites.google.com/a/dblab.postech.ac.kr/db/previous-resources
https://sites.google.com/a/dblab.postech.ac.kr/db/projects
https://sites.google.com/a/dblab.postech.ac.kr/db/projects/why-dbms
https://sites.google.com/a/dblab.postech.ac.kr/db/projects/redbase-ql
https://sites.google.com/a/dblab.postech.ac.kr/db/projects/redbase-1-paged-file
https://sites.google.com/a/dblab.postech.ac.kr/db/projects/redbase-1-paged-file/2-buffer-manager
https://sites.google.com/a/dblab.postech.ac.kr/db/projects/redbase-1-paged-file/object-manager
https://sites.google.com/a/dblab.postech.ac.kr/db/projects/redbase-1-paged-file/4-btree-manager
https://sites.google.com/a/dblab.postech.ac.kr/db/project-q-a
https://sites.google.com/a/dblab.postech.ac.kr/db/project-q-a/1-why-dbms
https://sites.google.com/a/dblab.postech.ac.kr/db/project-q-a/2-page-file
https://sites.google.com/a/dblab.postech.ac.kr/db/project-q-a/3-record-management
https://sites.google.com/a/dblab.postech.ac.kr/db/project-q-a/btree-manager
https://sites.google.com/a/dblab.postech.ac.kr/db/project-q-a/5-redbase-ql
```

Lab : Download PPTs with Selenium

Download PPTs

- Download all PPT files from the ‘Course Plan’ table using ‘selenium’

- **Course Plan**

Week	Topic
Week 1.	Introduction [PPT]
Week 2.	Disk and Files [PPT]
Week 3.	Indexing [PPT] [PPT2], Concurrent B-tree [PPT]
Week 4.	Recovery [PPT]
Week 5-7.	Relational algebra [PPT], Implementation of relational algebra (sort) [PPT]
Week 8.	Midterm exam
Week 9.	Query optimization [PPT]
Week 10.	SQL [PPT]
Week 11.	ER-model [PPT], Relational Model [PPT] [PPT]
Week 12.	Relational Calculus [PPT]
Week 13.	Database Design/Normalization [PPT]
Week 14.	New database technologies including Graph Databases [PPT]
Week 15.	Final Exam

Hint

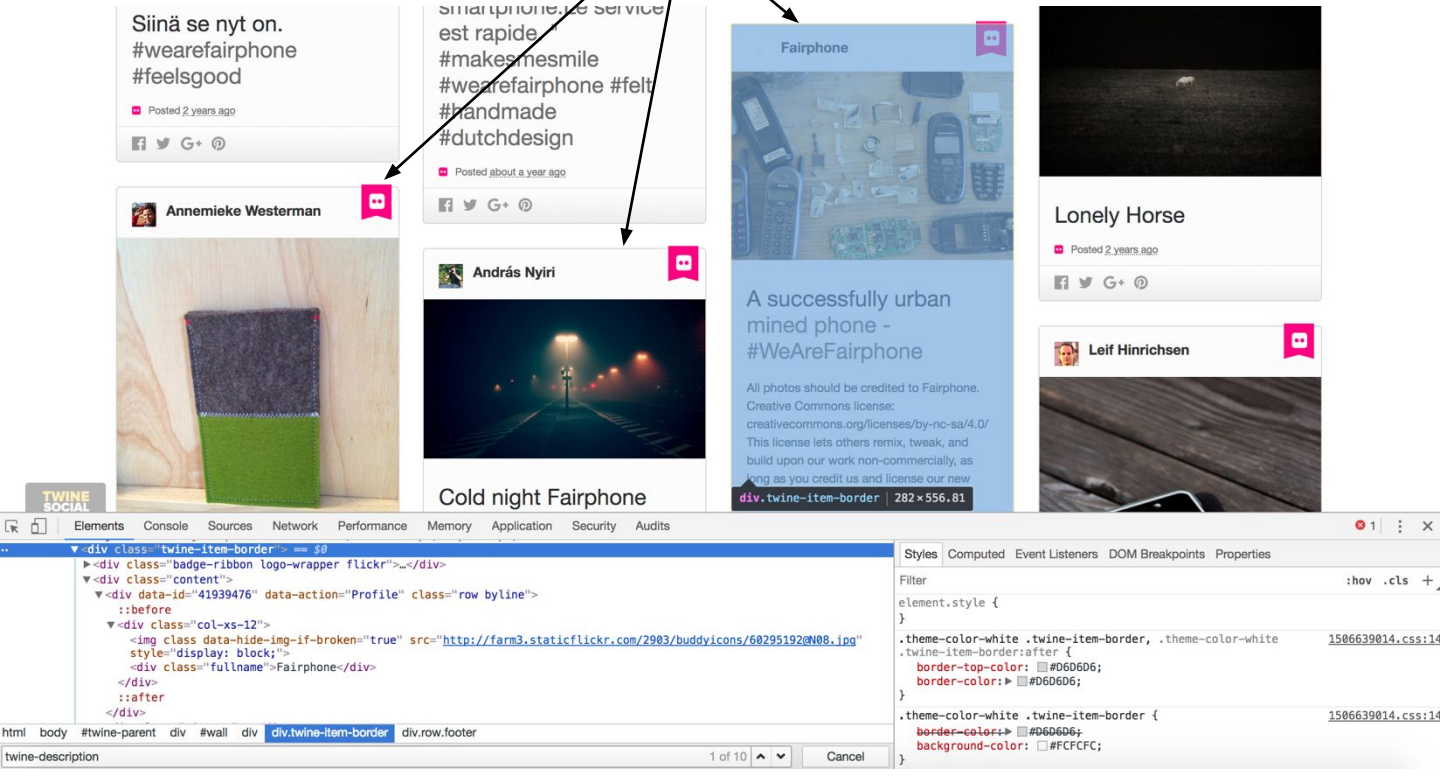
- Use XPath
- Refer to http://selenium-python.readthedocs.io/api.html (selenium.webdriver.remote.webelement.WebElement)

Selenium Example

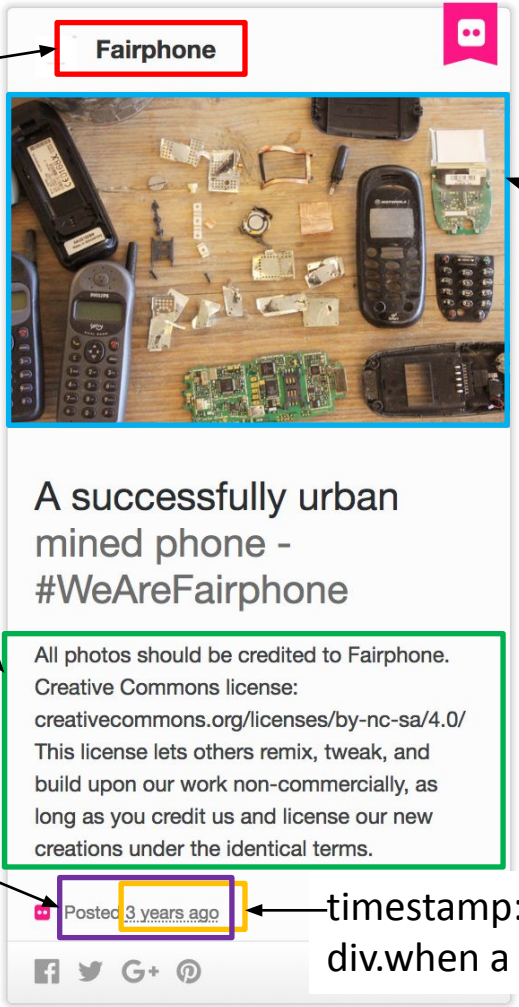
- Parse elements into the specific format
  - Utilize ‘Exception’ and ‘pass’ because all elements may not have the same structure

Selenium Example

- Extract data from each board (div.twine-item-border)



full_name:
div.fullname



picture:
div.picture img

text_content:
div.twine-description

original_link:
div.when a

timestamp:
div.when a abbr.timeago

```
from selenium.common.exceptions import NoSuchElementException, \
    WebDriverException
from selenium import webdriver


def find_text_element(html_element, element_css): ❶
    try:
        return html_element.find_element_by_css_selector(element_css).text ❷
    except NoSuchElementException:
        pass
    return None


def find_attr_element(html_element, element_css, attr): ❸
    try:
        return html_element.find_element_by_css_selector(
            element_css).get_attribute(attr) ❹
    except NoSuchElementException:
        pass
    return None


def get_browser():
    browser = webdriver.Firefox()
    return browser
```

1. find_text_element() takes an HTML element and CSS selector and return the text element
2. If there is no matched element, return None
3. find_attr_element() takes an HTML element, CSS selector, and the attribute
4. get_attribute() finds a matched attribute value

```
def main():
    browser = get_browser()
    browser.get('http://apps.twinesocial.com/fairphone')

    all_data = []
    browser.implicitly_wait(10) ❺
    try:
        all_bubbles = browser.find_elements_by_css_selector(
            'div.twine-item-border')
    except WebDriverException:
        browser.implicitly_wait(5)
        all_bubbles = browser.find_elements_by_css_selector(
            'div.twine-item-border')
    for elem in all_bubbles:
        elem_dict = {}
        content = elem.find_element_by_css_selector('div.content')
        elem_dict['full_name'] = find_text_element(
            content, 'div.fullname')
        elem_dict['short_name'] = find_attr_element(
            content, 'div.name', 'innerHTML')
        elem_dict['text_content'] = find_text_element(
            content, 'div.twine-description')
        elem_dict['timestamp'] = find_attr_element(
            elem, 'div.when a abbr.timeago', 'title') ❻
        elem_dict['original_link'] = find_attr_element(
            elem, 'div.when a', 'data-href')
        elem_dict['picture'] = find_attr_element(
            content, 'div.picture img', 'src')
        all_data.append(elem_dict)
    browser.quit() ❼
    return all_data ❽
```

5. implicitly_wait() let the browser to implicitly wait before moving to the next line of the code.
6. Passes CSS selectors to grab the title attribute of the abbr element located in an ‘a tag’ inside the ‘when div’
7. Closes the browser using the quit()
8. Returns the collected data

{'short_name': None, 'picture': 'http://farm6.staticflickr.com/5349/31064386195_f2b40567db_z.jpg', 'full_name': 'hyperlaxe (and far beyond)', 'text_content': None, 'original_link': 'http://www.flickr.com/photos/88018310@N00/31064386195', 'timestamp': '2016-11-17T20:31:56.000Z'}

{'short_name': '@Annemieke Westerman', 'picture': 'http://farm8.staticflickr.com/7393/26465226213_e826c09eb0_z.jpg', 'full_name': 'Annemieke Westerman', 'text_content': None, 'original_link': 'http://www.flickr.com/photos/28002699@N00/26465226213', 'timestamp': '2016-05-17T09:41:38.000Z'}

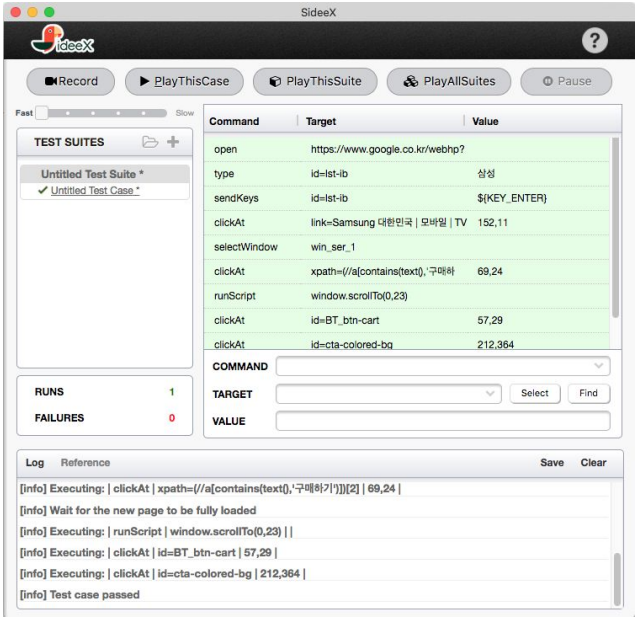
{'short_name': '@Annemieke Westerman', 'picture': 'http://farm2.staticflickr.com/1714/26489458071_efb12b96c7_z.jpg', 'full_name': 'Annemieke Westerman', 'text_content': None, 'original_link': 'http://www.flickr.com/photos/28002699@N00/26489458071', 'timestamp': '2016-04-21T09:00:35.000Z'}

{'short_name': None, 'picture': 'http://farm2.staticflickr.com/1599/24026097961_3ed3d80dac_z.jpg', 'full_name': 'samelisivonen', 'text_content': None, 'original_link': 'http://www.flickr.com/photos/25781323@N07/24026097961', 'timestamp': '2016-01-01T18:10:40.000Z'}

{'short_name': '@mrdvent', 'picture': 'http://farm1.staticflickr.com/531/18949494933_af38e2ac22_z.jpg', 'full_name': 'Martin Parkes', 'text_content': None, 'original_link': 'http://www.flickr.com/photos/133636392@N07/18949494933', 'timestamp': '2015-07-10T06:24:36.000Z'}

{'short_name': None, 'picture': 'http://farm9.staticflickr.com/8610/16039060834_aac9eb544f_z.jpg', 'full_name': 'Fairphone', 'text_content': 'All photos should be credited to Fairphone. Creative Commons license: creativecommons.org/licenses/by-nc-sa/4.0/ This license lets others remix, tweak, and build upon our work non-commercially, as long as you credit us and license our new creations under the identical terms.', 'original_link': 'http://www.flickr.com/photos/60295192@N08/16039060834', 'timestamp': '2015-02-27T10:28:30.000Z'}

- Record-playback automated web application testing tool based on Selenium



Spidering the Web

- Spider is a robot that crawls pages and follows rules to identify good content or more pages to follow
- If you need to capture data from more than one page on a site, a spider is the best solution

Scrapy

- Popular Python web spider
- Uses LXML and Twisted (asynchronous network engine)
- Built-in features
  - Export results in several formats (CSV, JSON, etc)
  - Easy-to-use server deployment structure to run multiple on-demand scrapers

Spider types

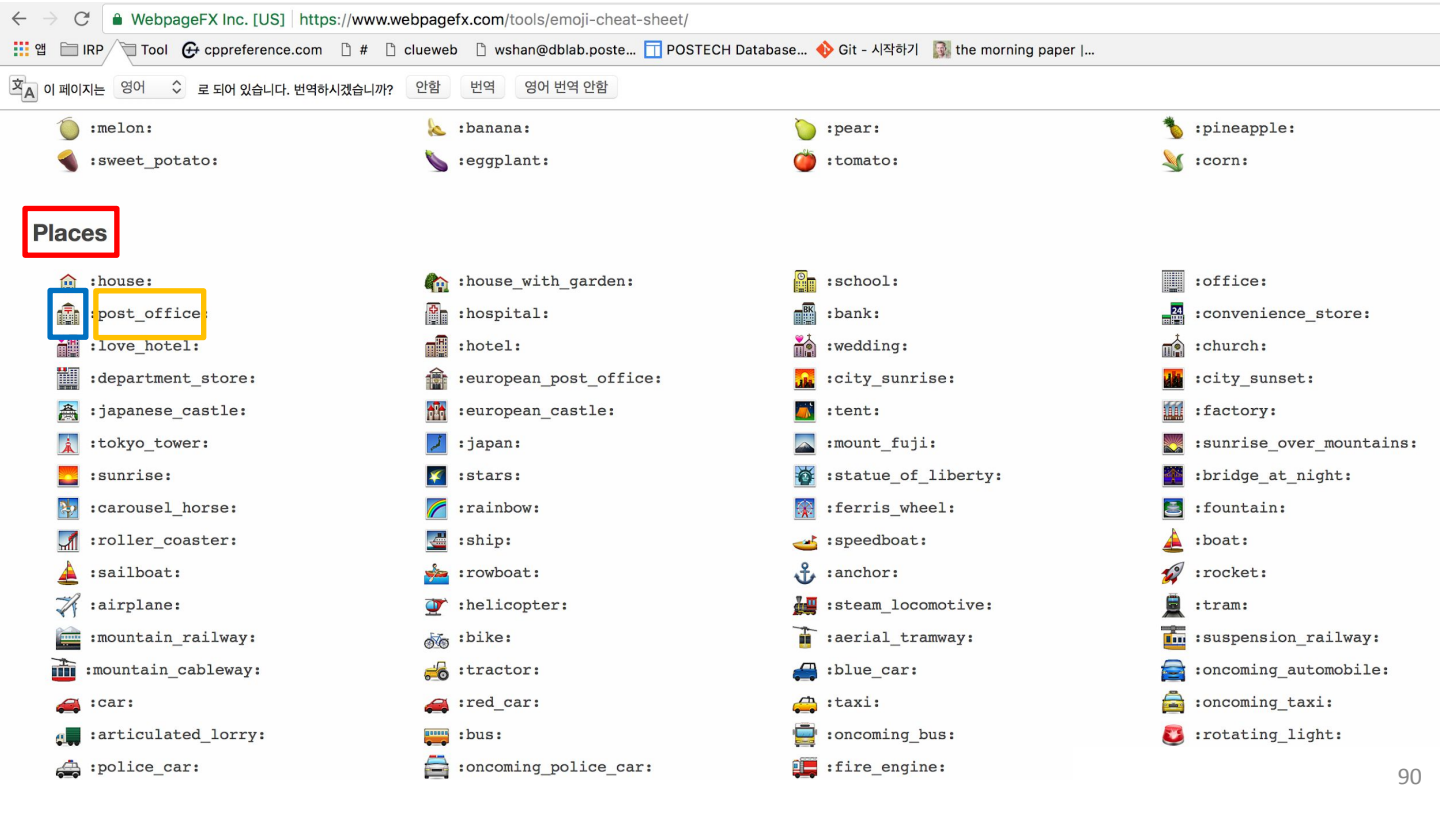
- Built-in Spider types

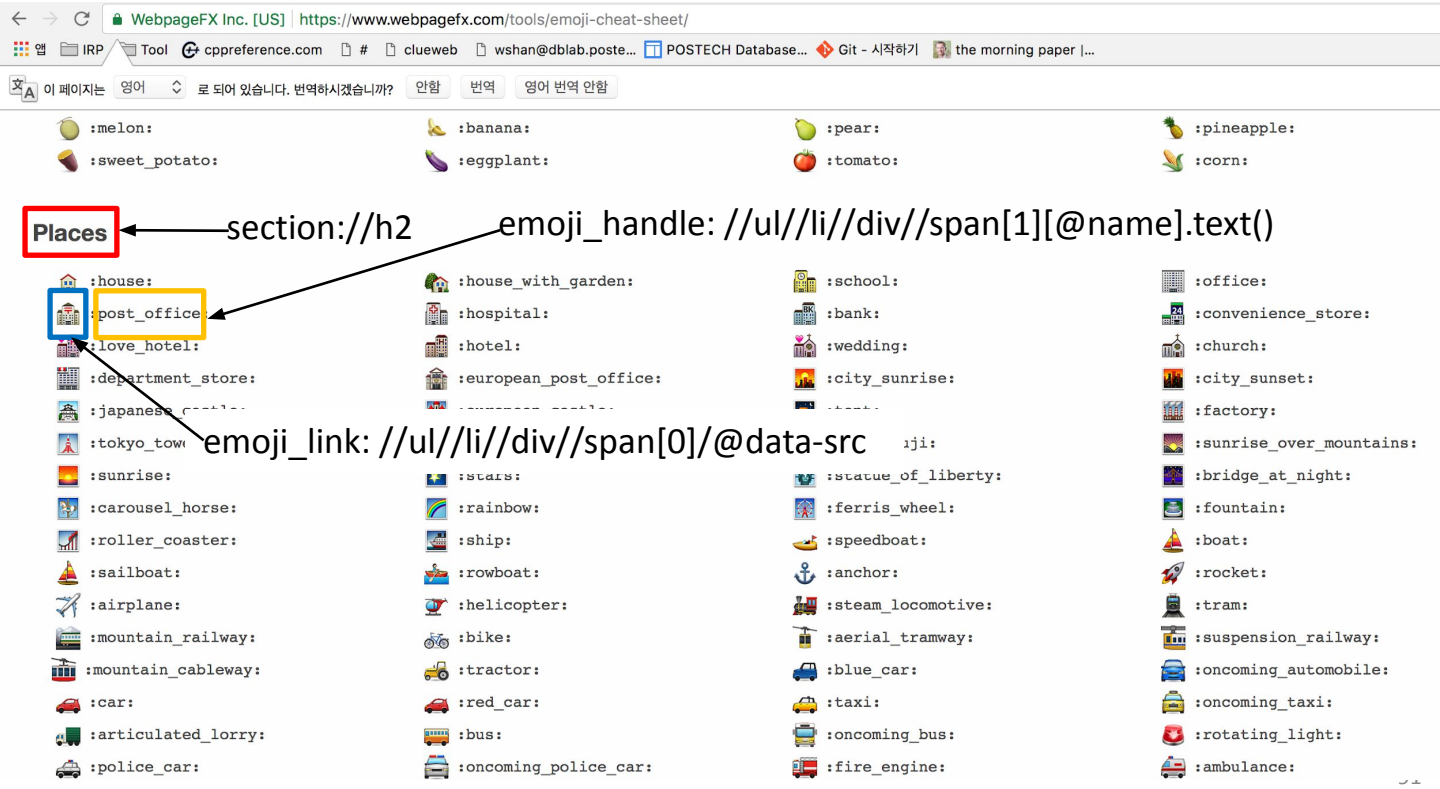
Spider name	Main purpose
Spider	Used to normal web scraping
Crawl Spider	Used to parse a domain using the given a set of regex rules on how to follow links and identify good pages
XMLFeed Spider	Used to parse XML feeds (like RSS) and pull contents from XML nodes
CSVFeed Spider	Used to parse CSV feeds (or URLs) and pull contents from rows
SiteMap Spider	Used to parse site maps for a given list of domains

- Extended spider types can be defined by inheritance

Example using Spider for a single page

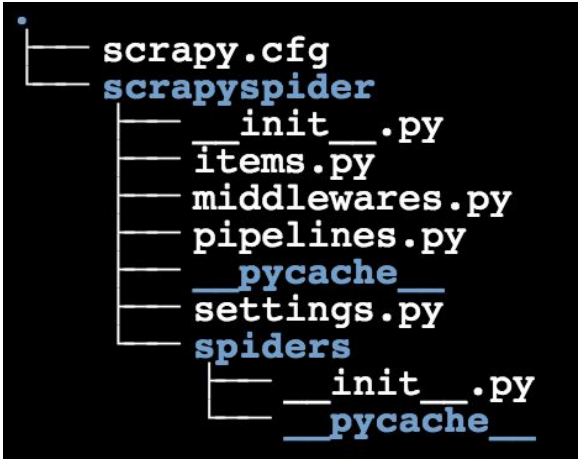
Build a spider to extract categories, images and names of emoticons.




WebpageFX Inc. [US] | https://www.webpagefx.com/tools/emoji-cheat-sheet/
앱 IRP Tool cppreference.com # clueweb wshan@dblab.poste... POSTECH Database... Git - 시작하기 the morning paper |...
이 페이지는 영어 로 되어 있습니다. 번역하시겠습니까? 안함 번역 영어 번역 안함
:melon: :banana: :pear: :pineapple:
:sweet_potato: :eggplant: :tomato: :corn:
Places
section://h2
emoji_handle: //ul//li//div//span[1][@name].text()
:house: :house_with_garden: :school: :office:
:post_office: :hospital: :bank: :convenience_store:
:love_hotel: :hotel: :wedding: :church:
:department_store: :european_post_office: :city_sunrise: :city_sunset:
:factory:
:tokyo_tow :sunrise_over_mountains:
emoji_link: //ul//li//div//span[0]//@data-src
:sunrise: :statue_of_liberty: :bridge_at_night:
:carousel_horse: :rainbow: :ferris_wheel: :fountain:
:roller_coaster: :ship: :speedboat: :boat:
:sailboat: :rowboat: :anchor: :rocket:
:airplane: :helicopter: :steam_locomotive: :tram:
:mountain_railway: :bike: :aerial_tramway: :suspension_railway:
:mountain_cableway: :tractor: :blue_car: :oncoming_automobile:
:car: :red_car: :taxi: :oncoming_taxi:
:articulated_lorry: :bus: :oncoming_bus: :rotating_light:
:police_car: :oncoming_police_car: :fire_engine: :ambulance:

1. Execute scrapy to create a bunch of folders including items.py

   ex) scrapy startproject scrapyspider



2. Define user-defined items by using the built-in scrapy.Item class

ex)

```
import scrapy

class EmojiSpiderItem(scrapy.Item):
    emoji_handle = scrapy.Field()
    emoji_link = scrapy.Field()
    section = scrapy.Field()
```

3. Modify a spider file in the ‘spiders’ folder

ex)

```
import scrapy
from scrapyspider.items import EmojiSpiderItem①

class EmoSpider(scrapy.Spider):②
    name = 'emo'③
    allowed_domains = ['emoji-cheat-sheet.com']④
    start_urls = [
        'http://www.emoji-cheat-sheet.com/',⑤
    ]
```

1. You can import your defined class for data collecting
2. We inherit simple scrapy.Spider class to create class ‘EmoSpider’. We should define which URLs to scrape and what to do with scraped content
3. Define a spider name what we will use in command-line
4. If a link that found is not included in this list, it will be ignored. If you don’t want to scrape specific domain, it will be useful
5. The spider class uses the ‘start_urls’ to iterate through a listing of URLs to scrape

4. Redefine parse() function to fill our item's attributes, 'section', 'emoji_link', 'emoji_handle'

```
def parse(self, response):
    headers = response.xpath('//h2|//h3')
    lists = response.xpath('//ul')
    all_items = []①
    for header, list_cont in zip(headers, lists):
        section = header.xpath('text()').extract()[0]②
        for li in list_cont.xpath('li'):
            item = EmojiSpiderItem()③
            item['section'] = section
            spans = li.xpath('div/span')
            if len(spans):
                link = spans[0].xpath('@data-src').extract()④
                if link:
                    item['emoji_link'] = response.url + link[0]⑤
                handle_code = spans[1].xpath('text()').extract()
            else:
                handle_code = li.xpath('div/text()').extract()
            if handle_code:
                item['emoji_handle'] = handle_code[0]⑥
            all_items.append(item)⑦
    return all_items⑧
```

1. Creates list to contain items
2. The extract() method returns a list of matching elements. In order to select item, we only get a first element of the list
3. Creates item object using our defined item class
4. To extract data attributes, uses the XPath @ selector
5. If there is a 'data-src' attribute, we use it to create link with the basic url
6. Sets the 'emoji_handle' field
7. Appends the new item to our final result
8. Returns the list of all found items. Scrapy will use a returned item to proceed with the scraping usually by saving, cleaning the data

5. Execute ‘Scrapy crawl emo’

ex) If you want to save results in a file, try this command.
‘scrapy crawl emo -o items.csv’

```
2019-06-09 15:40:29 [scrapy.core.scraper] DEBUG: Scraped from
    <200 https://www.webfx.com/tools/emoji-cheat-sheet/>
    {'emoji_handle': '/play butts',
     'section': 'Campfire also supports a few sounds'}
2019-06-09 15:40:29 [scrapy.core.engine] INFO: Closing spider (finished)
2019-06-09 15:40:29 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
    {'downloader/request_bytes': 1152,
     'downloader/request_count': 5,
     'downloader/request_method_count/GET': 5,
     'downloader/response_bytes': 52949,
     'downloader/response_count': 5,
     'downloader/response_status_count/200': 3,
     'downloader/response_status_count/301': 2,
     'finish_reason': 'finished',
     'finish_time': datetime.datetime(2019, 6, 9, 6, 40, 29, 863982),
     'item_scraped_count': 925,
     'log_count/DEBUG': 931,
     'log_count/INFO': 8,
     'response_received_count': 3,
     'scheduler/dequeued': 2,
     'scheduler/dequeued/memory': 2,
     'scheduler/enqueued': 2,
     'scheduler/enqueued/memory': 2,
     'start_time': datetime.datetime(2019, 6, 9, 6, 40, 26, 604701)}
2019-06-09 15:40:29 [scrapy.core.engine] INFO: Spider closed (finished)
```

6. Check the output, ‘items.csv’

items.csv

```
emoji_handle,emoji_link,section

bowtie,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/bowtie.png,People

smile,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/smile.png,People

simple_smile,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/simple_smile.png,People

laughing,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/laughing.png,People

blush,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/blush.png,People

smiley,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/smiley.png,People

relaxed,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/relaxed.png,People

smirk,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/smirk.png,People

heart_eyes,https://www.webfx.com/tools/emoji-cheat-sheet/graphics/emojis/heart_eyes.png,People
```

Fuzzy Matching

- Fuzzy String Matching (Approximate String Matching) is the process of finding strings that approximatively match a given pattern

  ex) spell checking, matching DNA sequences, spam filtering

- ‘fuzzywuzzy’ is a Python library for Fuzzy String Matching

Fuzzy Matching

- Uses ratio() or partial_ratio() to calculate a similarity score between two strings

```
from fuzzywuzzy import fuzz

fuzz.ratio('Barack Obama', 'Barack H. Obama')
# 89
fuzz.partial_ratio('Barack Obama', 'Barack H. Obama')
# 75

fuzz.ratio('Barack H Obama', 'Barack H. Obama')
# 97
fuzz.partial_ratio('Barack H Obama', 'Barack H. Obama')
# 92
```

The method partial_ratio() is good for the case when someone has forgotten a word. But, it may not be able to distinguish two strings with different meanings but similar spelling. For example, 'does' and 'doesn't'.

1. The method ratio() returns similarity of the sequencing of the strings
2. The method partial_ratio() returns similarity of the sequencing of the closest matching substrings

Fuzzy Matching

- Uses token(or word) as unit for matching. It is useful when the order of tokens doesn't change the meaning

```
fuzz.token_sort_ratio('Barack Obama', 'Barack H. Obama')
# 92
fuzz.token_set_ratio('Barack Obama', 'Barack H. Obama')
# 100

fuzz.token_sort_ratio('Barack H Obama', 'Barack H. Obama')
# 100
fuzz.token_set_ratio('Barack H Obama', 'Barack H. Obama')
# 100
```

1. The method token_sort_ratio() allows us to match strings despite word order. It first sort each string and then compare
2. The method token_set_ratio() compares sets of the tokens to see intersection and difference

Fuzzy Matching

- Uses ‘process’ module in case we have a list of options and when we want to find the closest match(es)

There are only four choices.

```
from fuzzywuzzy import process

choices = ['Yes', 'No', 'Maybe', 'N/A']

process.extract('ya', choices, limit=2) ❶

process.extractOne('ya', choices) ❷

process.extract('nope', choices, limit=2)

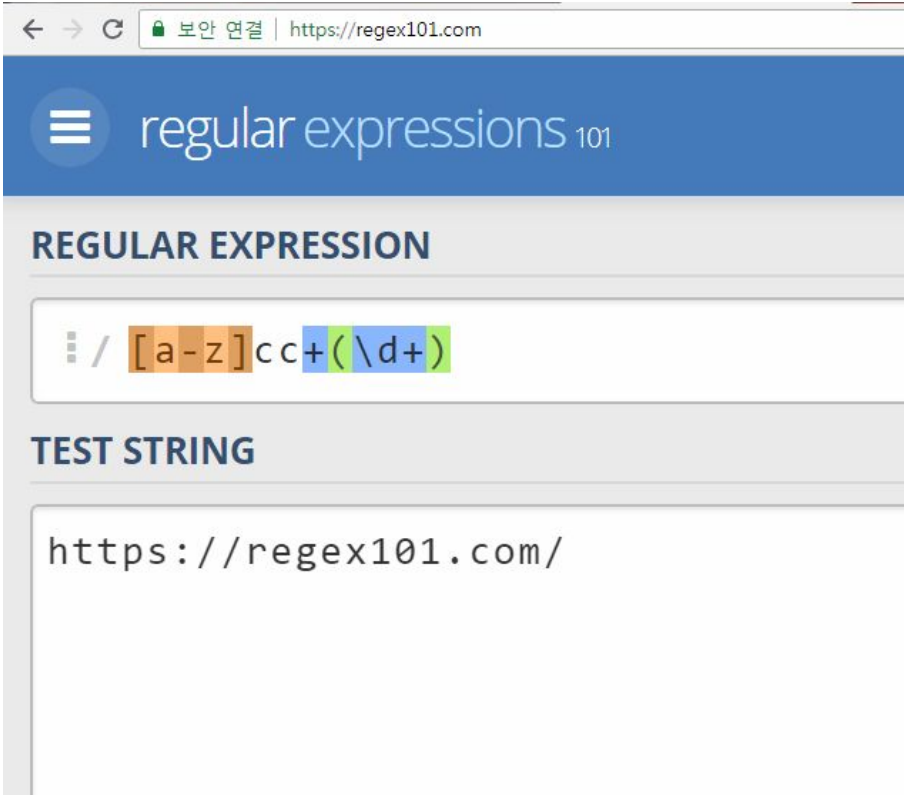
process.extractOne('nope', choices)
```

1. The method extract() compare strings to the list of possible matches. It returns two possible matches from the variable choices
2. The method extractOne() only returns the best match

RegEx Matching

- Regular expressions used to search for specific elements, or groups of elements, that match a pattern

Regular Expressions



Matching Multiple Characters

- Can match sets of characters, or multiple and more elaborate sets and sequences of characters:
- Match the character ‘a’: `a`
- Match the character ‘a’, ‘b’, or ‘c’: `[abc]`
- Match any character except ‘a’, ‘b’, or ‘c’: `[^abc]`
- Match any digit: `\d` (= `[0123456789]` or `[0-9]`)
- Match any alphanumeric: `\w` (= `[a-zA-Z0-9_]`)
- Match any whitespace: `\s` (= `[ \t\n\r\f\v]`)
- Match any character: `.`
- Special characters must be escaped: `.^$*+?{}\[]|()`

Thanks to: Zico Kolter

Matching sequences and repeated characters

- A few common modifiers (available in Python and most other high-level languages; +, {n}, {n,} *may* not):
  - Match character ‘a’ exactly once: `a`
  - Match character ‘a’ zero or once: `a?`
  - Match character ‘a’ zero or more times: `a*`
  - Match character ‘a’ one or more times: `a+`
  - Match character ‘a’ exactly *n* times: `a{n}`
  - Match character ‘a’ at least n times: `a{n,}`
- Example: match all instances of “University of <somewhere>” where <somewhere> is an alphanumeric string with at least 3 characters:
  - `\s*University\sof\s\w{3,}`

Basic methods

```
import re

# Find the index of the 1st occurrence of "dblab"
match = re.search(r"dblab", text)
print( match.start() )
```

```
# Does start of text match "dblab"?
match = re.match(r"dblab", text)
```

```
# Iterate over all matches for "dblab" in text
for match in re.finditer(r"dblab", text):
    print( match.start() )
```

```
# Return all matches of "dblab" in the text
match = re.findall(r"dblab", text)
```

Compiled Regexes

- If you're going to reuse the same regex many times, or if you aren't but things are going slowly for some reason, try compiling the regular expression.

```
# Compile the regular expression "dblab"
regex = re.compile(r"dblab")

# Use it repeatedly to search for matches in text
regex.match( text )     # does start of text match?
regex.search( text )    # find the first match or None
regex.findall( text )   # find all matches
```

Matching with multiple groups

- Let's see if we can create some patterns where we need to reference more than one group

```
import re

name_regex = '([A-Z]\w+) ([A-Z]\w+)' ❶

names = "Barack Obama, Ronald Reagan, Nancy Drew"

name_match = re.match(name_regex, names) ❷

name_match.group()

name_match.groups() ❸

name_regex = '(?P<first_name>[A-Z]\w+) (?P<last_name>[A-Z]\w+)' ❹

for name in re.finditer(name_regex, names): ❺
    print 'Meet {}!'.format(name.group('first_name')) ❻
```

1. We use the same capital word pattern twice with parentheses. Parentheses are used to define groups
2. We use pattern with two regex groups in the method match()
3. The method groups() return a list of all the matches
4. If we naming each group, it is more clear to write the code
5. The method finditer() is similar to the method findall(), but we can iterate over the string with iterator
6. Using the formatting and naming information of the group, we can print our data in the format we want

Thank you

Q&A