

# [301] Web 3

Tyler Caraza-Harter

# Learning Objectives Today

Use BeautifulSoup module

- prettify, find\_all, find, get\_text

Learn about scraping

- Document Object Model
- extracting links
- robots.txt

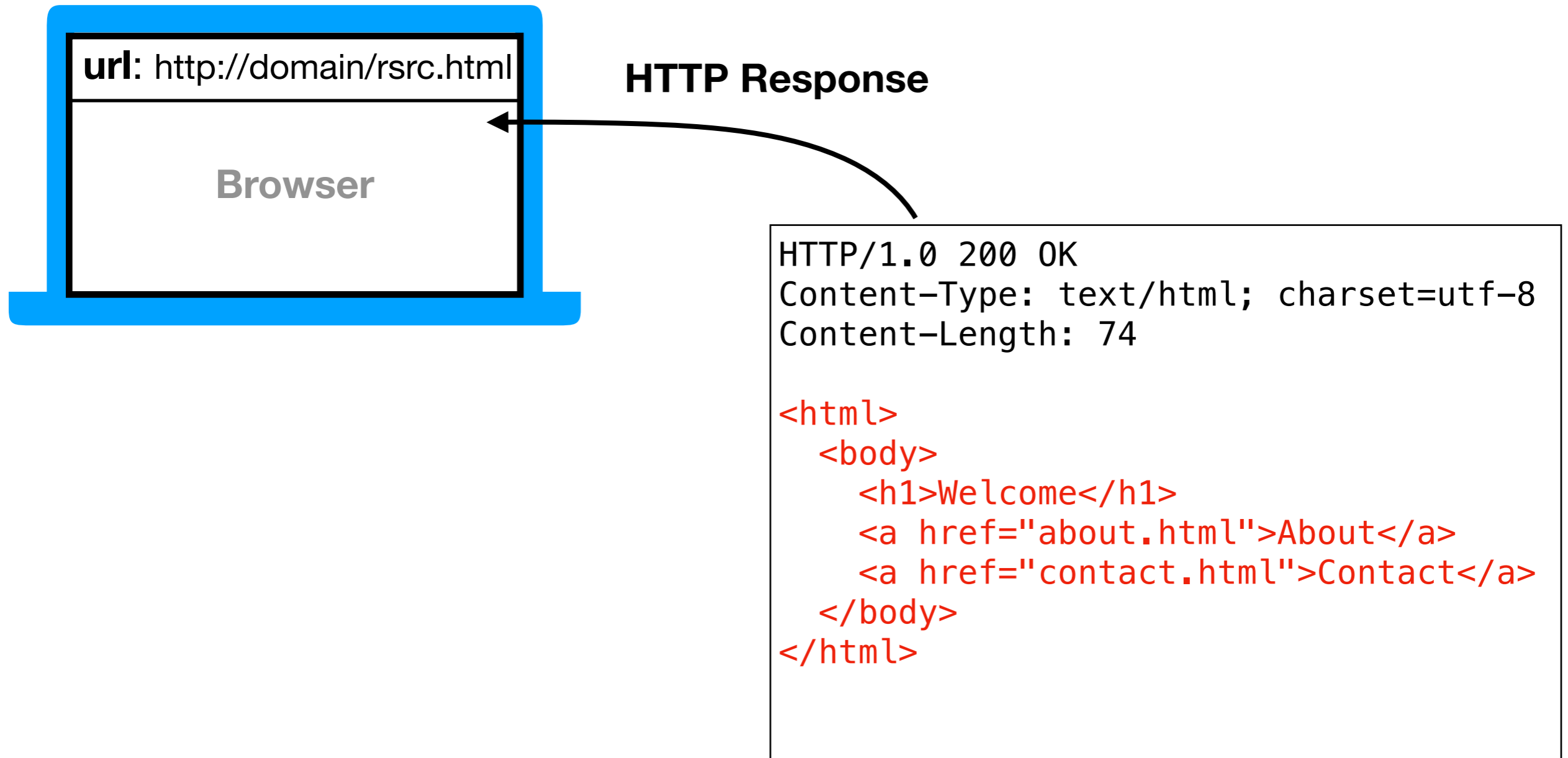
# Outline

Document Object Model

BeautifulSoup module

Scraping States from Wikipedia

# What does a web browser do when it gets some HTML in an HTTP response?



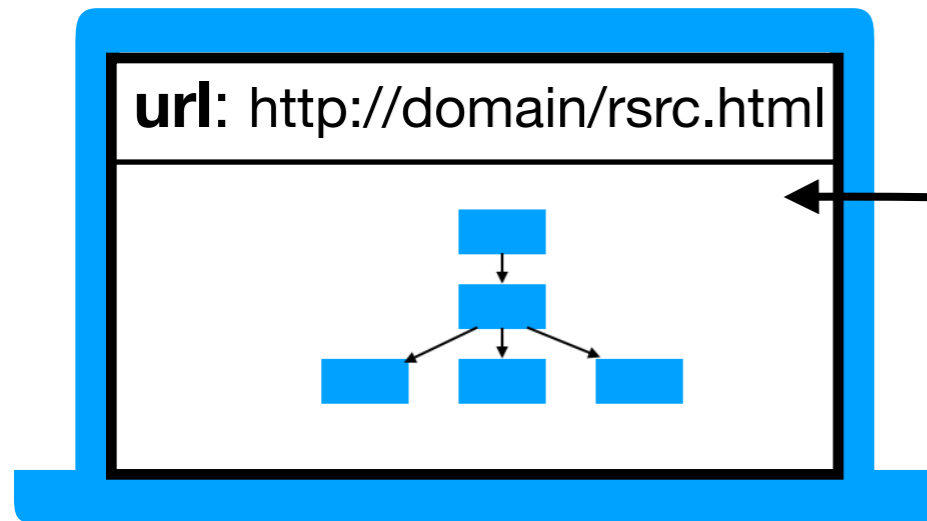
**url:** http://domain/rsrc.html

```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```

## HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```



## HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```

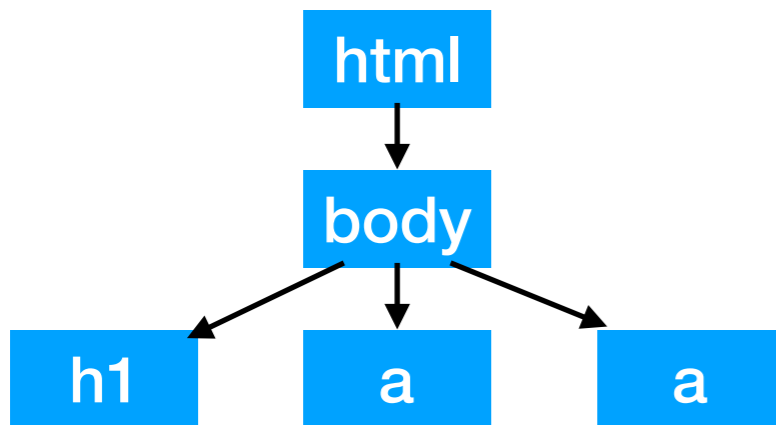
before displaying a page, the browser uses HTML to generate a Document Object Model (DOM Tree)

url: http://domain/rsrc.html

HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

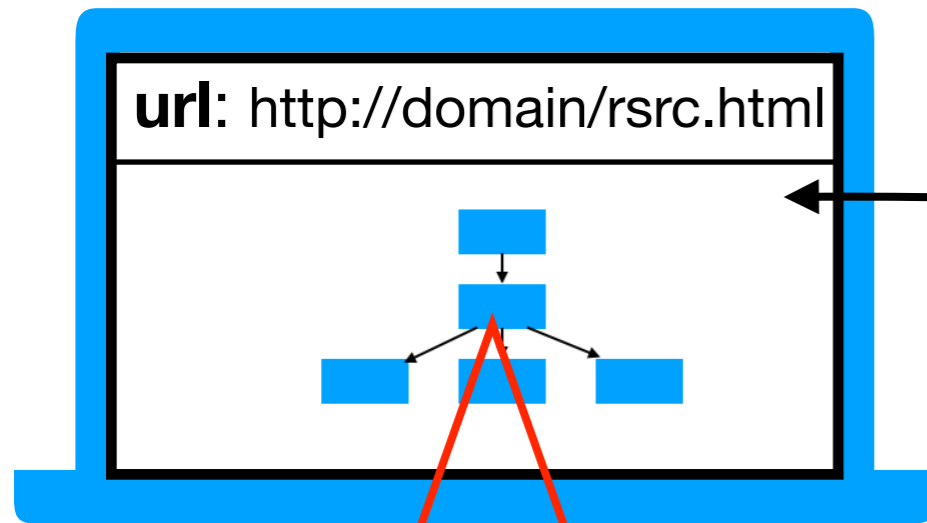
```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```



vocab: elements

# Elements may contain

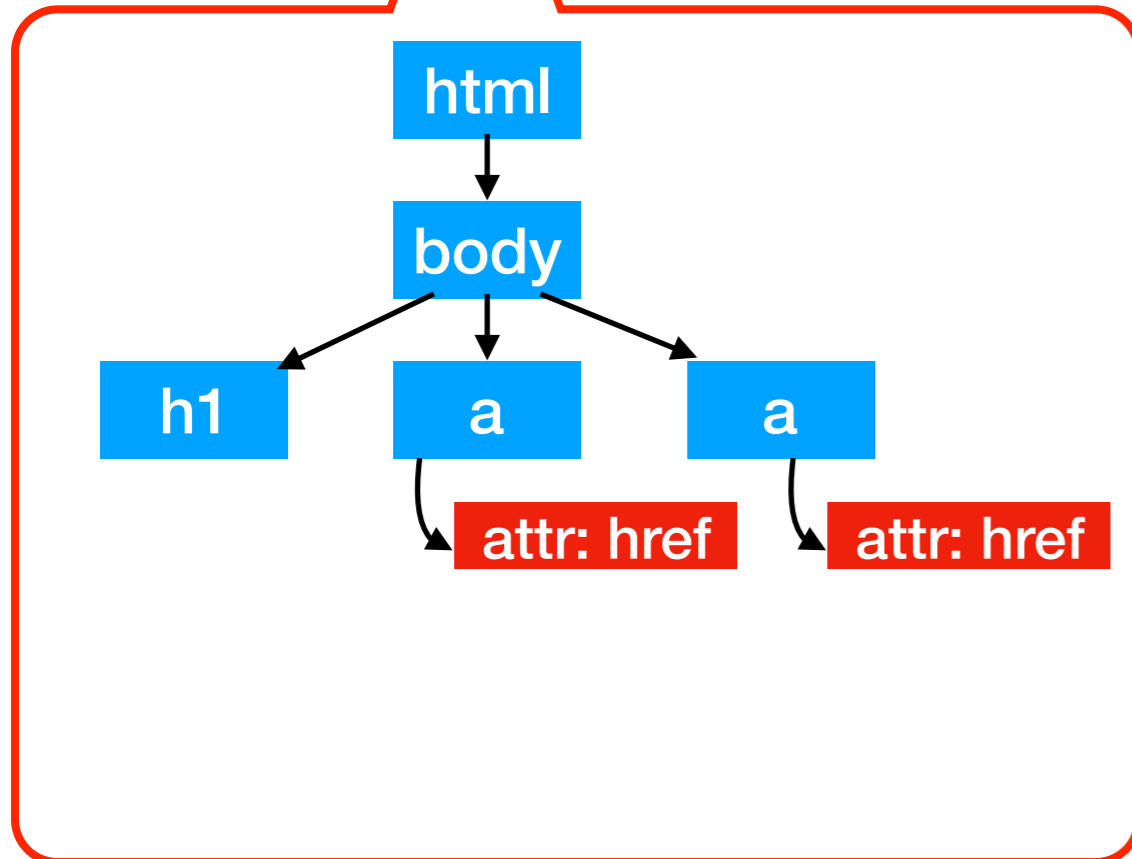
- attributes



HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

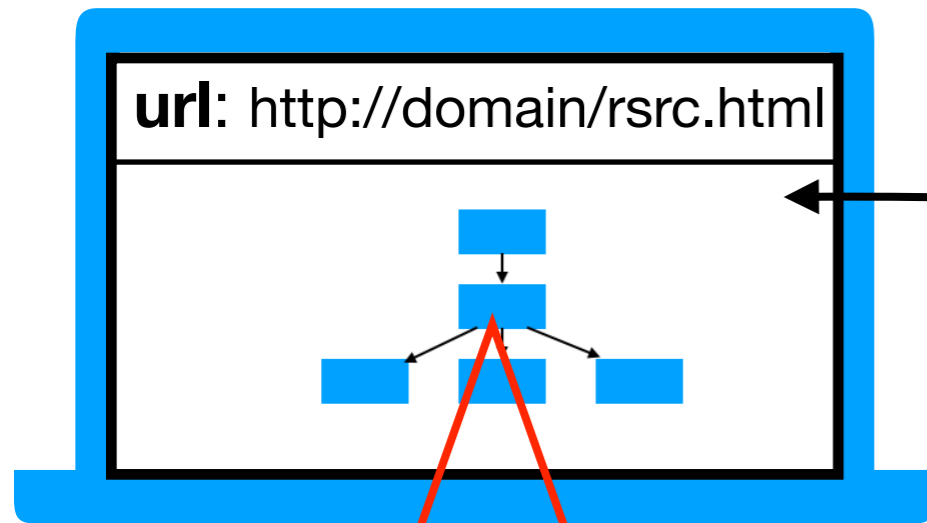
```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```





## Elements may contain

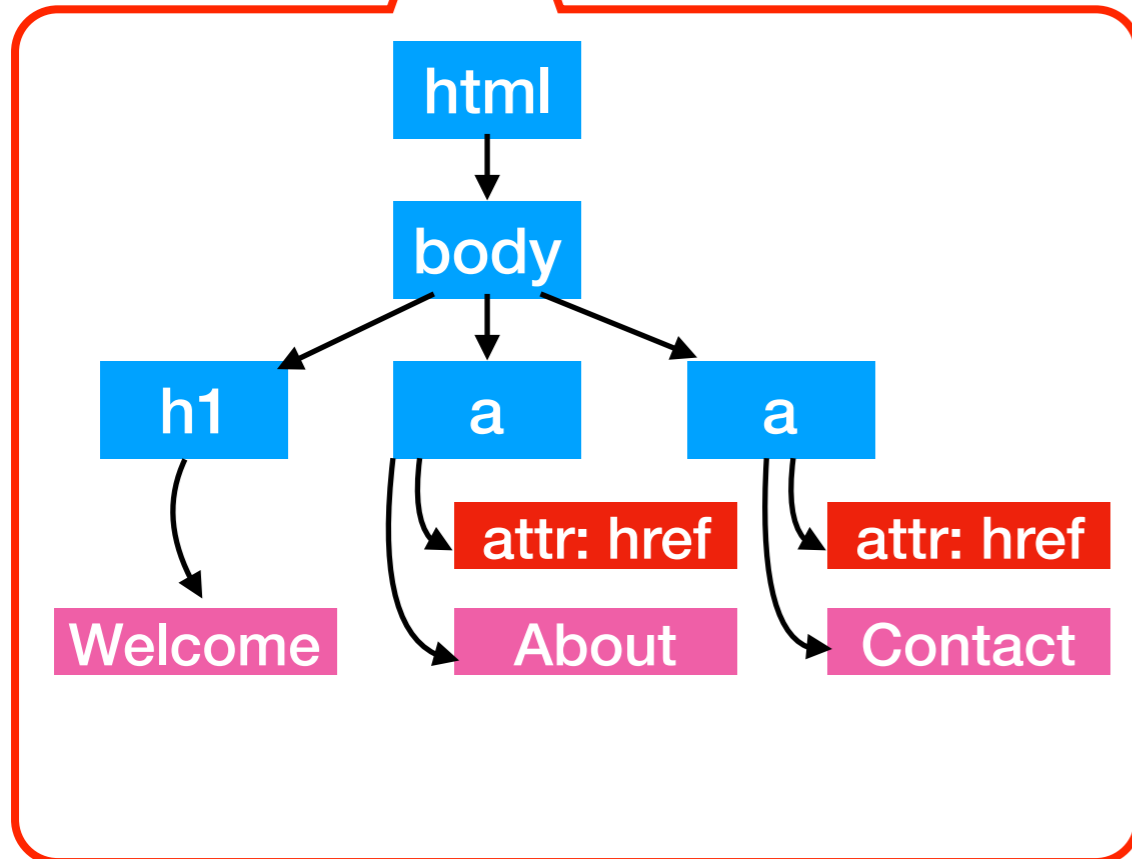
- attributes
- text



HTTP Response

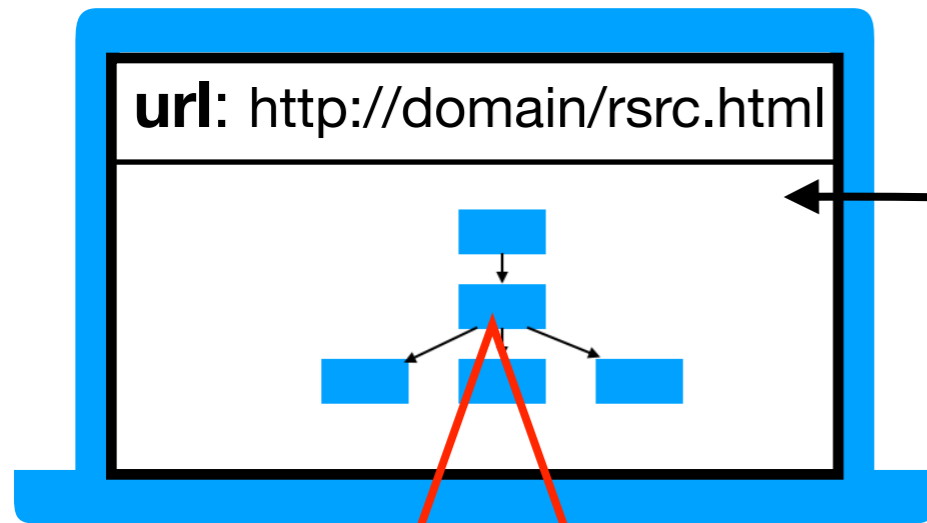
```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```



## Elements may contain

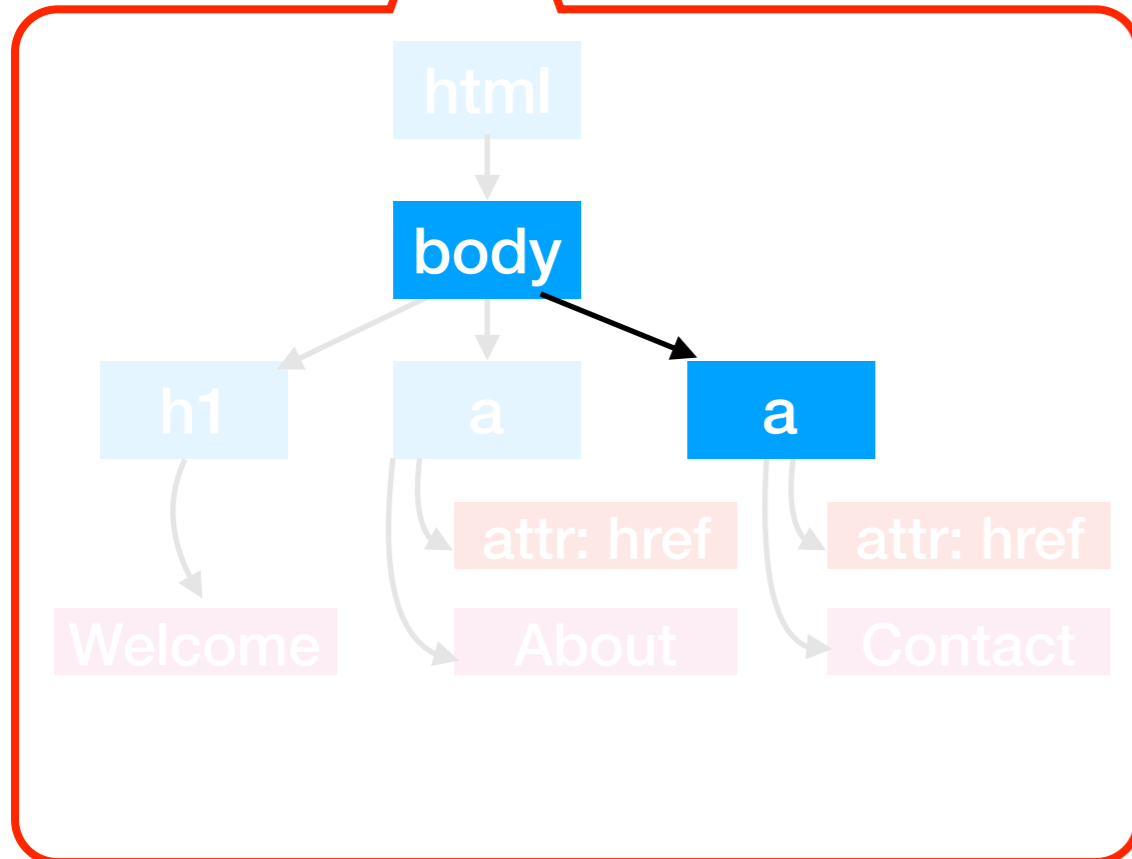
- attributes
- text
- other **elements**



**HTTP Response**

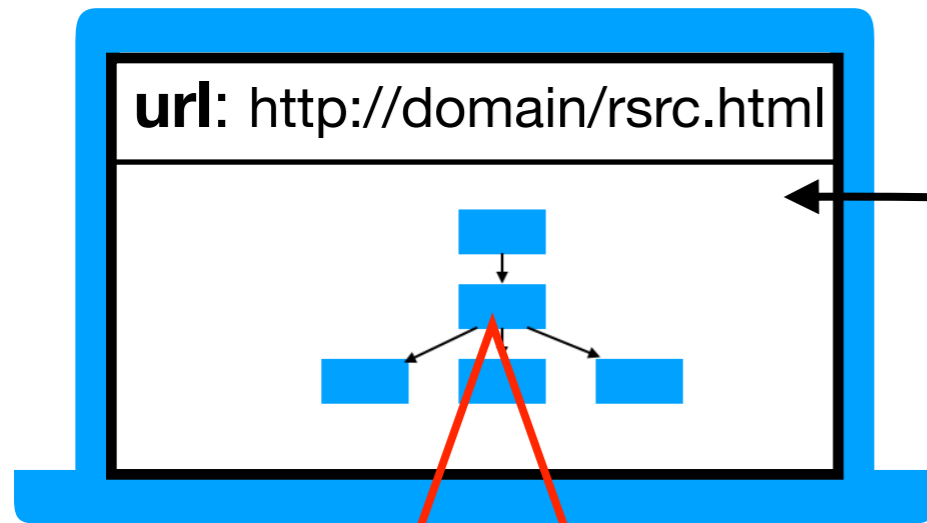
```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```



## Elements may contain

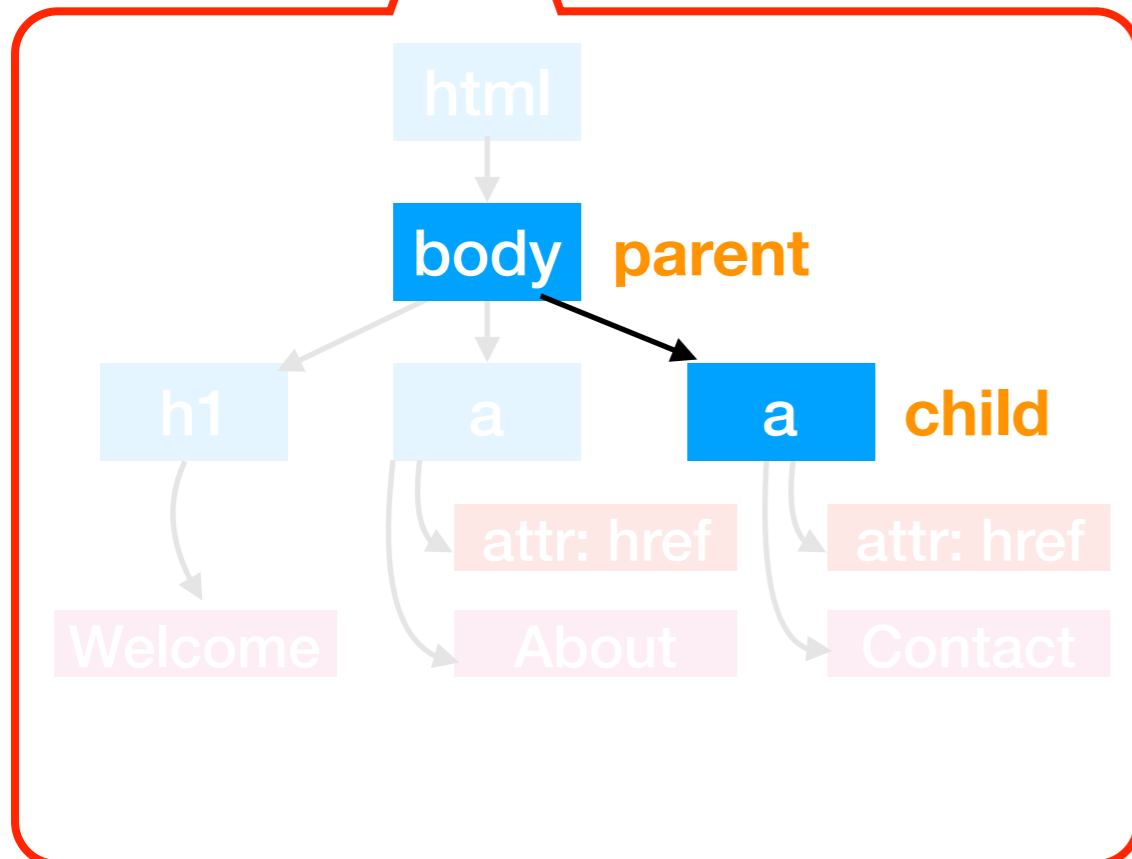
- attributes
- text
- other **elements**



HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```



## Elements may contain

- attributes
- text
- other **elements**



HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74

<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```

browser renders (displays)  
the DOM tree

Python program gets back the same info  
as a web browser (HTTP and HTML)

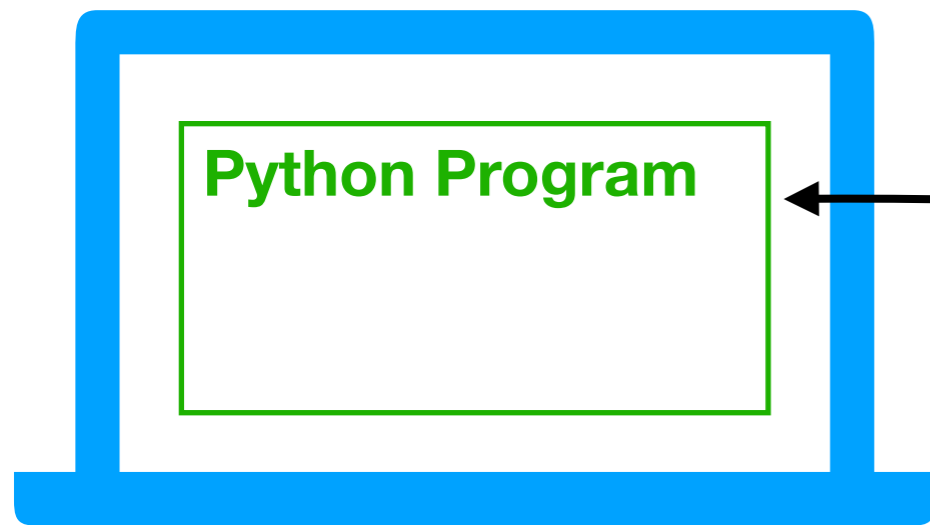


**HTTP Response**

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74

<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```

Depending on application, we may want to use:  
**1. HTTP information**



HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```

**Example:** determine whether page exists

```
r = requests.get(...)
code = r.status_code
...
```

Depending on application, we may want to use:

1. HTTP information
2. **raw HTML (or JSON, CSV, etc)**



HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
```

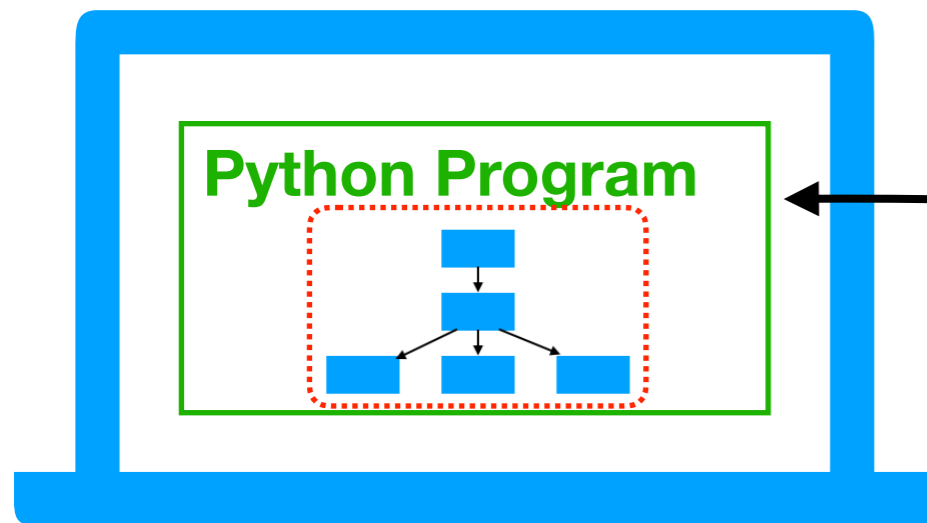
```
<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```

**Example:** downloader

```
r = requests.get(...)
f = open(..., "w")
f.write(r.text)
f.close()
```

Depending on application, we may want to use:

1. HTTP information
2. raw HTML (or JSON, CSV, etc)
3. **model of HTML document**



**Example:** extract URLs  
from every hyperlink

```
from bs4 import BeautifulSoup
# parse HTML to a model.
# TODAY's topic...
```

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74

<html>
  <body>
    <h1>Welcome</h1>
    <a href="about.html">About</a>
    <a href="contact.html">Contact</a>
  </body>
</html>
```



# Outline

Document Object Model

BeautifulSoup module

Scraping States from Wikipedia

# BeautifulSoup module

## Purpose

- convert HTML (downloaded from the web or otherwise) to a model of **elements**, **attributes**, and **text**
- simple functions for searching for elements for a particular type (e.g., find all "a" tags to extract all hyperlinks)

## Installation

- `pip install beautifulsoup4`

## Using it

- `from bs4 import BeautifulSoup`

# Parsing HTML

new type



```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

## Items

- x
- y
- z

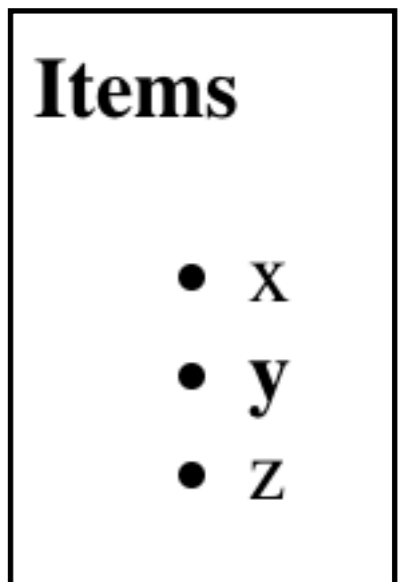
# Parsing HTML

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

this could have come from anywhere:

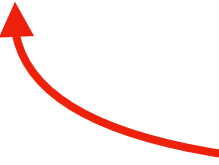
- hardcoded string
- something from requests GET
- loaded from local file



# Parsing HTML

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```



we'll always use this  
(other strings parse  
other formats)

## Items

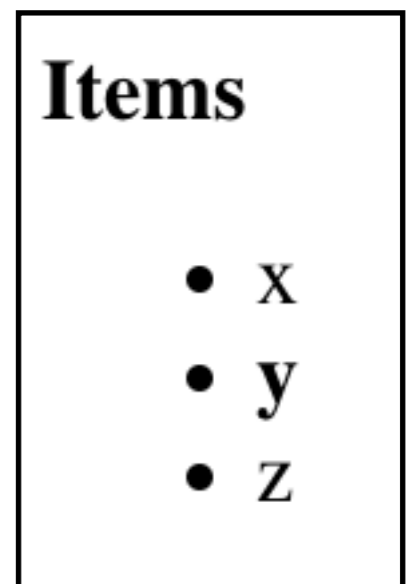
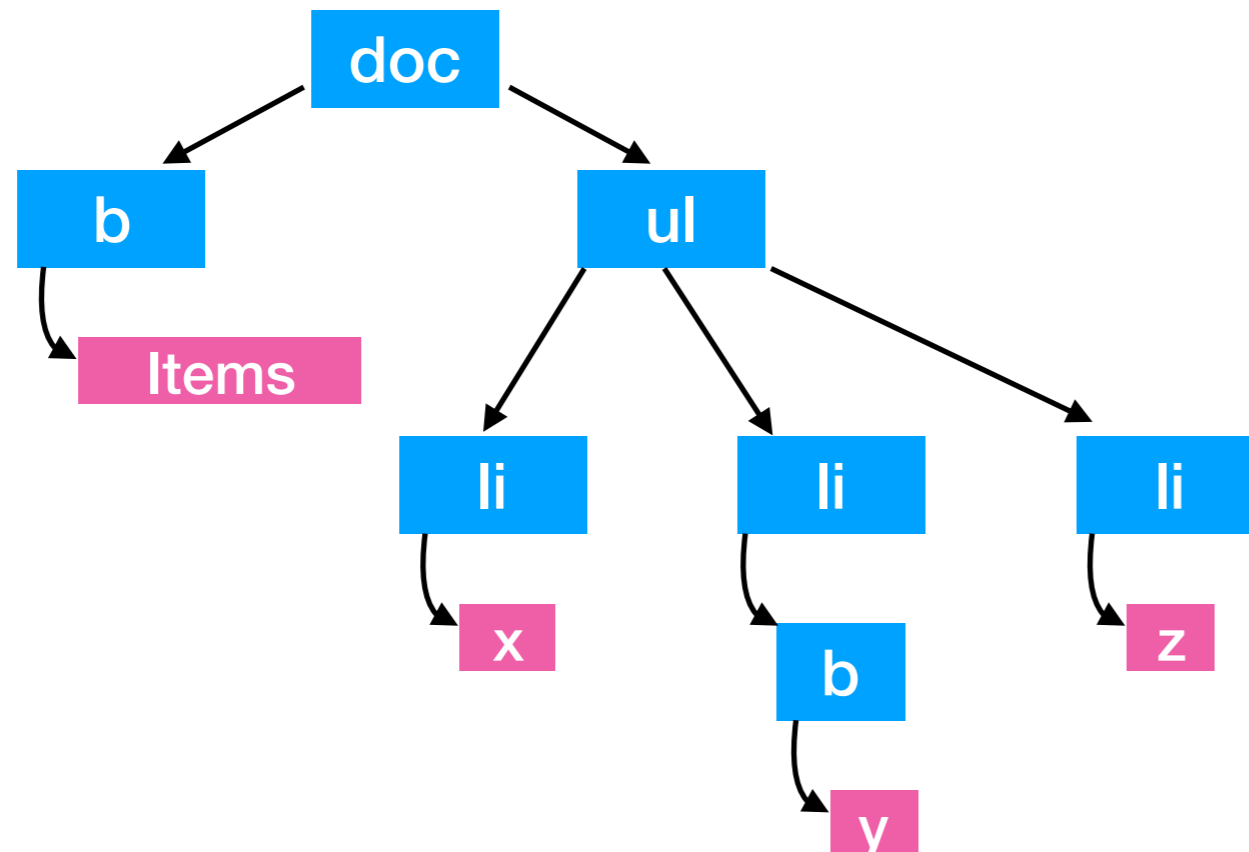
- x
- **y**
- z

# Parsing HTML

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

document object that  
we can easily analyze



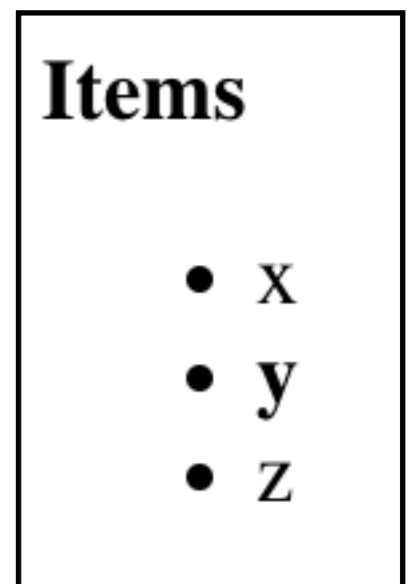
# Parsing HTML

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
print(doc.pretty())
```

```
<b>  
  Items  
</b>  
<ul>  
  <li>  
    x  
  </li>  
  <li>  
    <b>  
      y  
    </b>  
  </li>  
  <li>  
    z  
  </li>  
</ul>
```

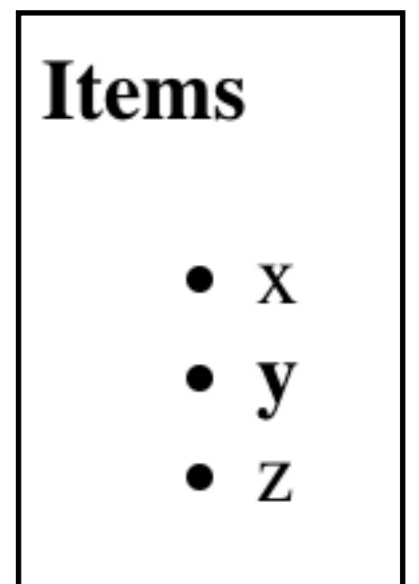
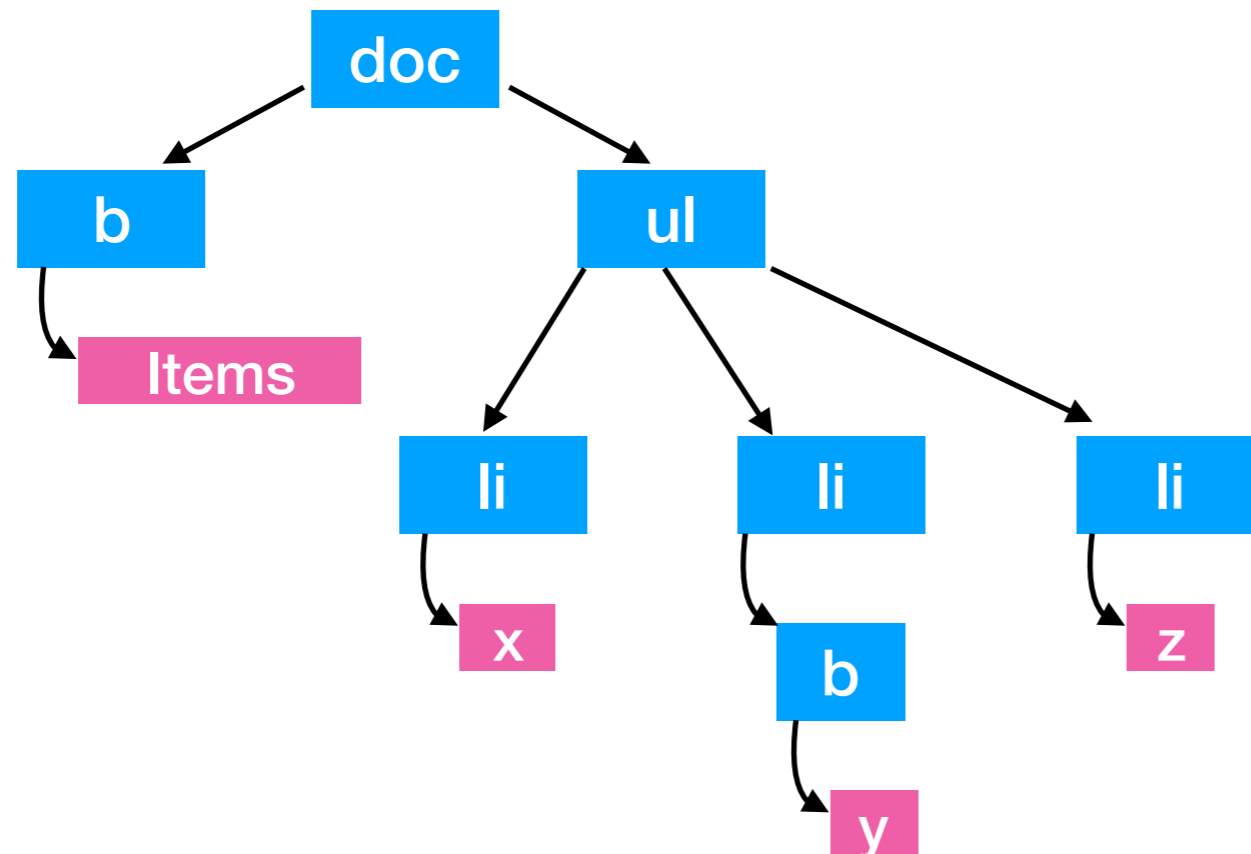


# Searching for Elements

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("li")  
print(len(elements))
```



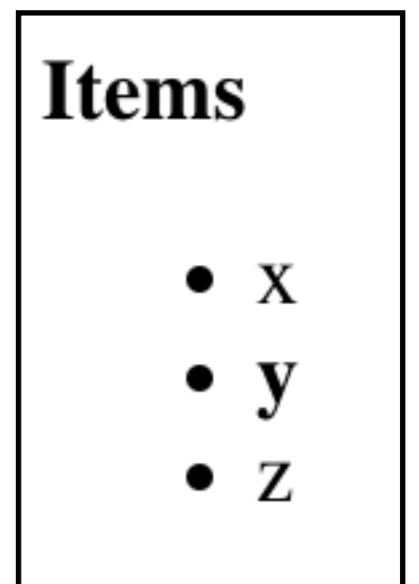
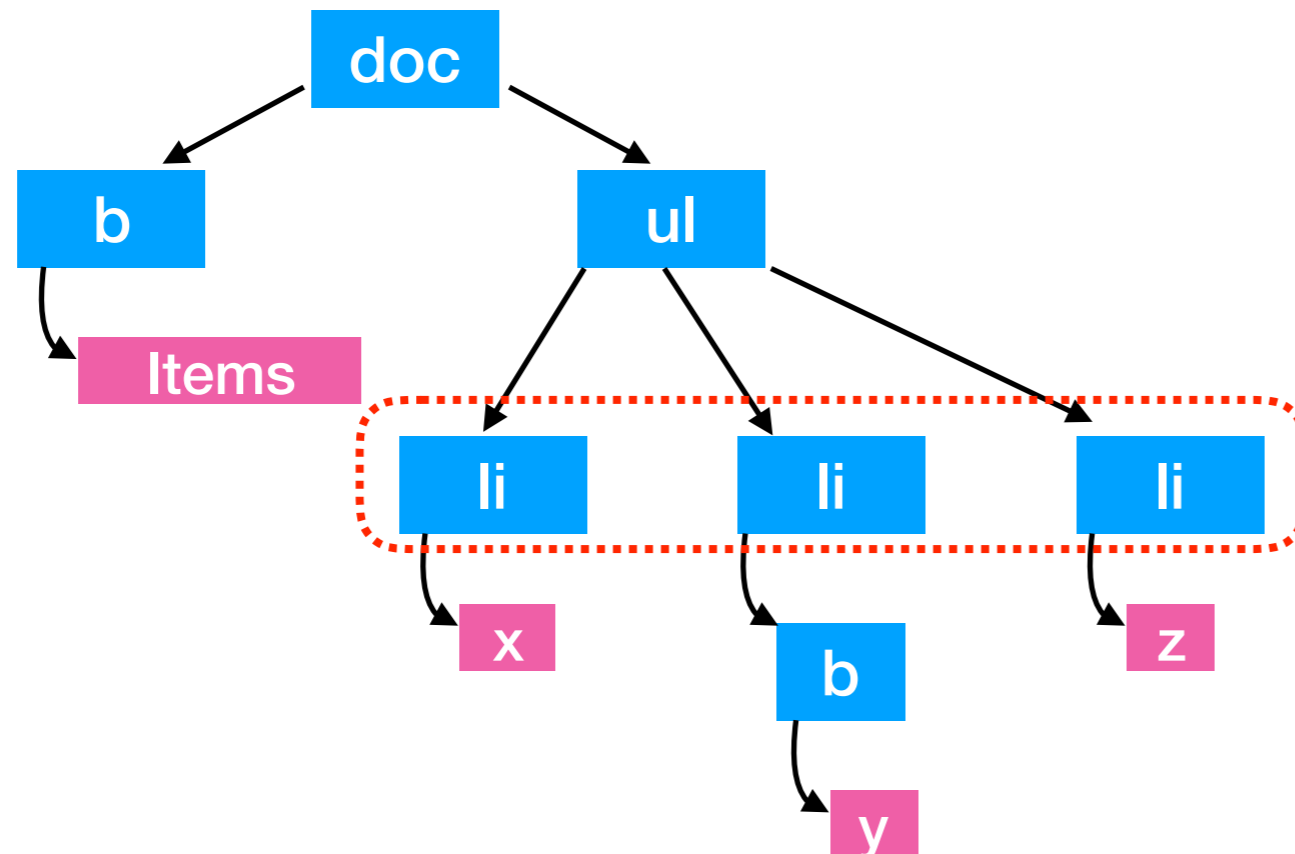


# Searching for Elements

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("li")  
print(len(elements))
```

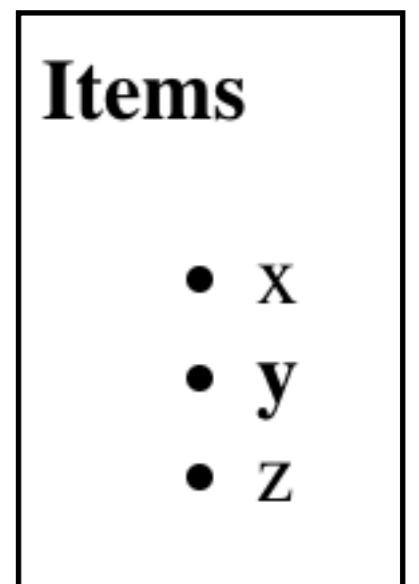
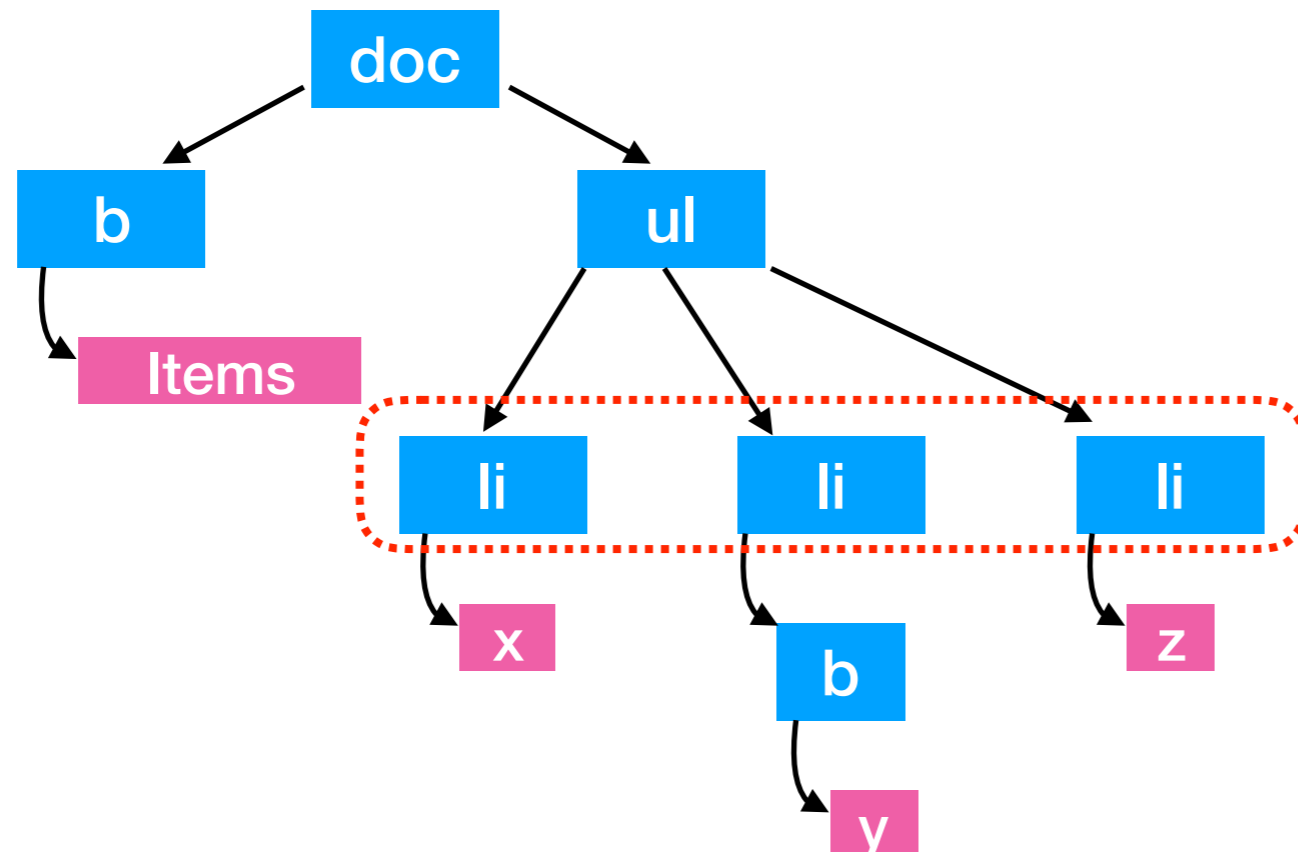


# Searching for Elements

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("li")    list of three elements  
print(len(elements))           prints 3
```

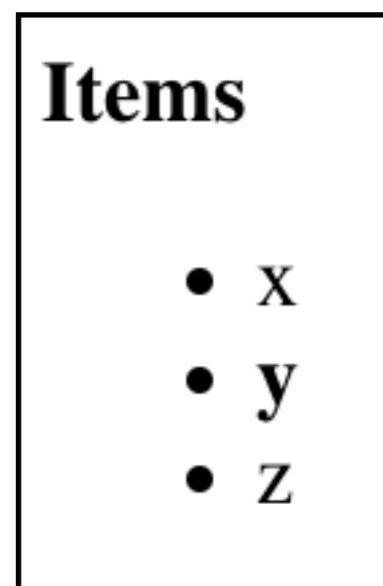
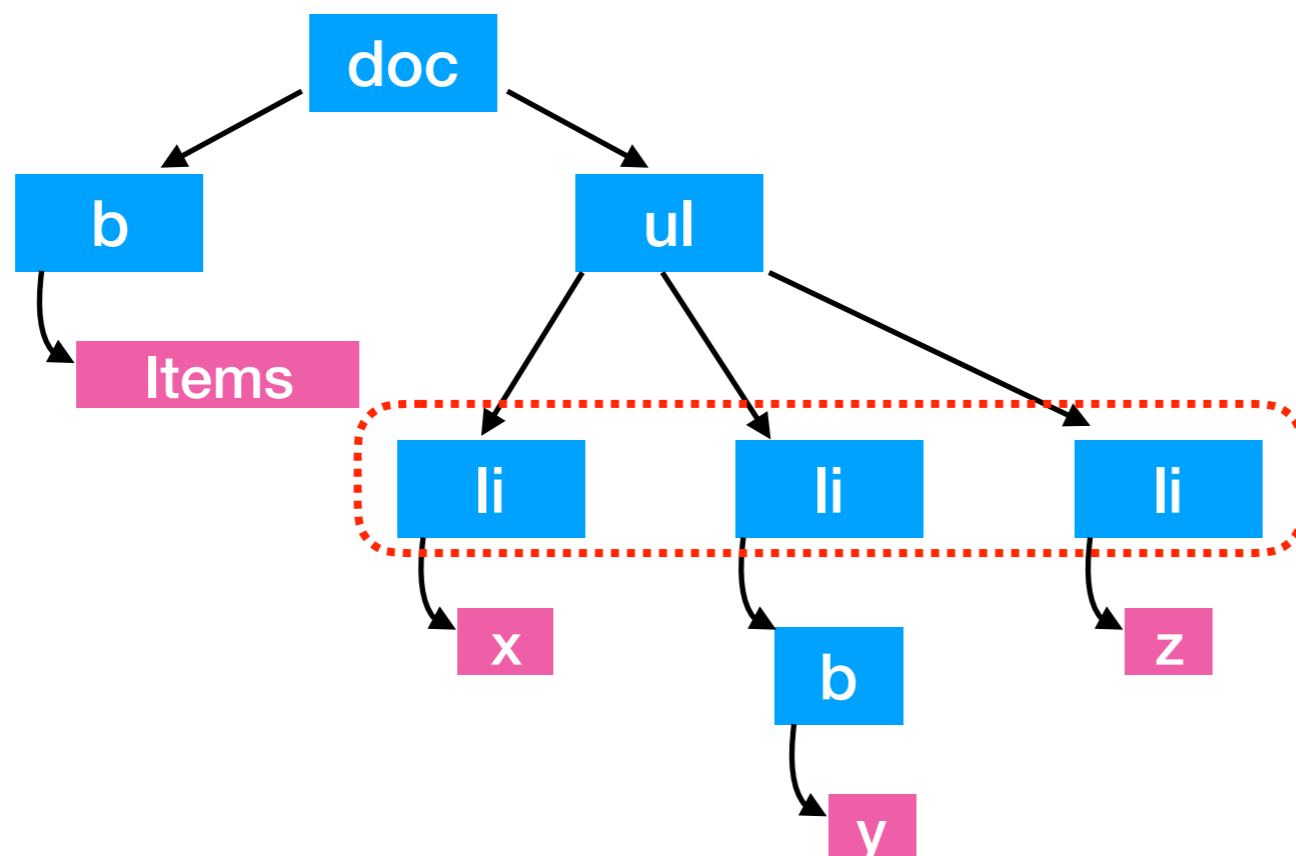


# Searching for Elements

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("li")    list of three elements  
print(len(elements))            prints 3
```



# Extracting Text

```
from bs4 import BeautifulSoup
```

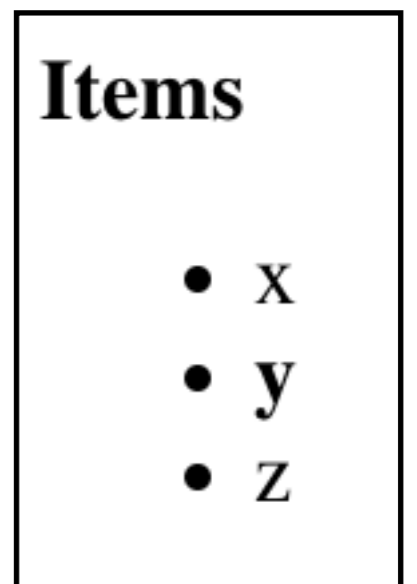
```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("li")  
print(len(elements))
```

```
for e in elements:  
    print(e.get_text())
```

## Prints:

x  
y  
z



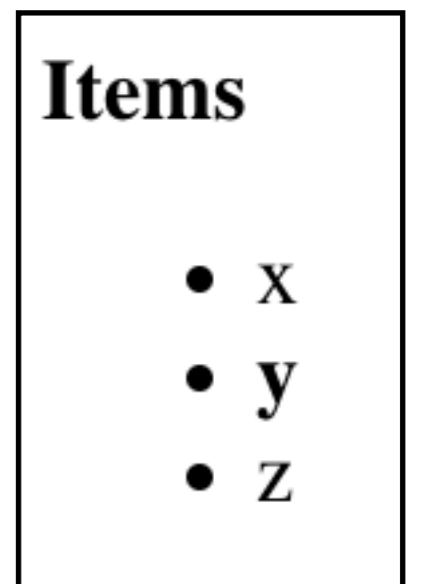
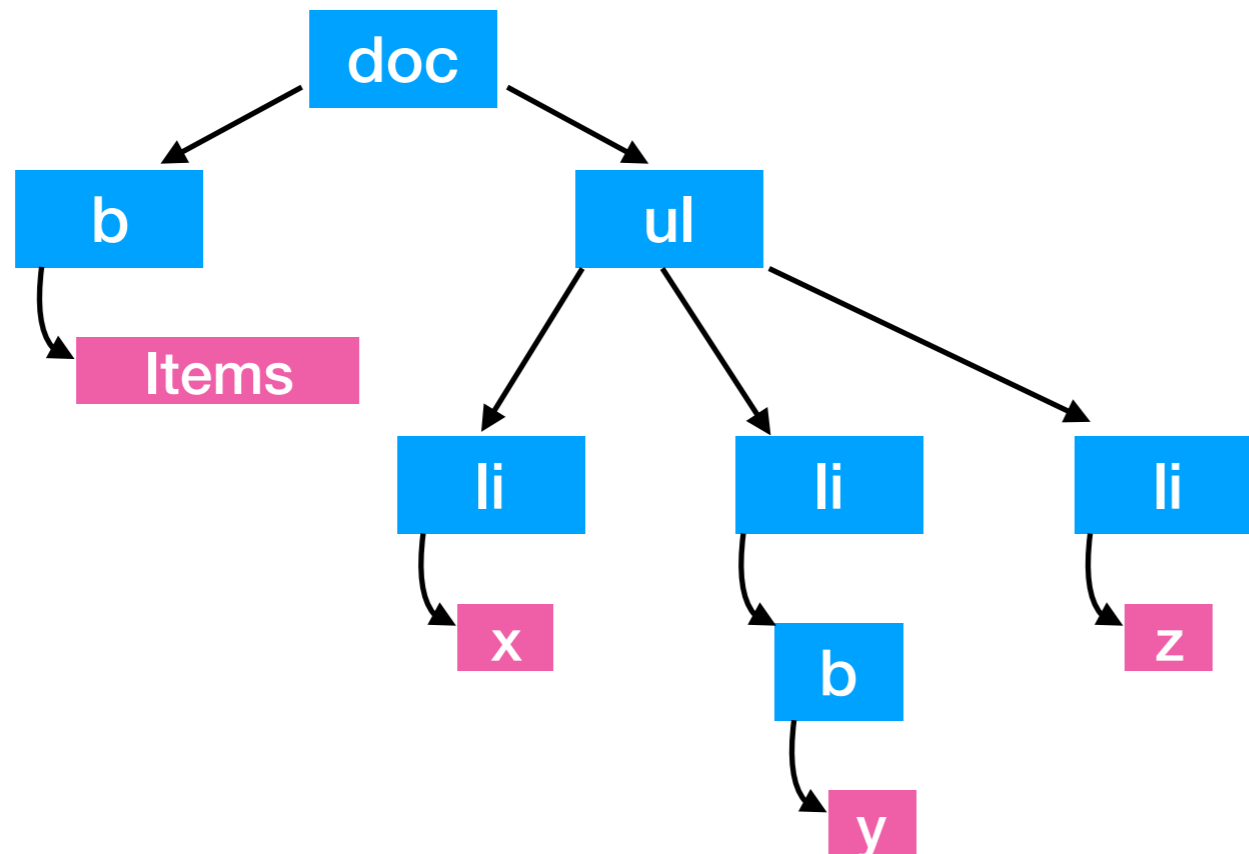
# Parsing HTML

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("b")  
print(len(elements))
```

now look for all bold elements



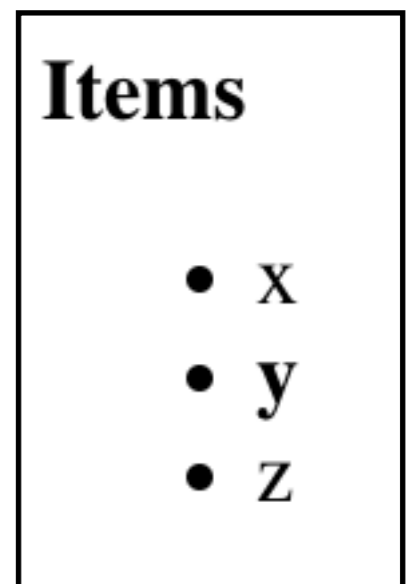
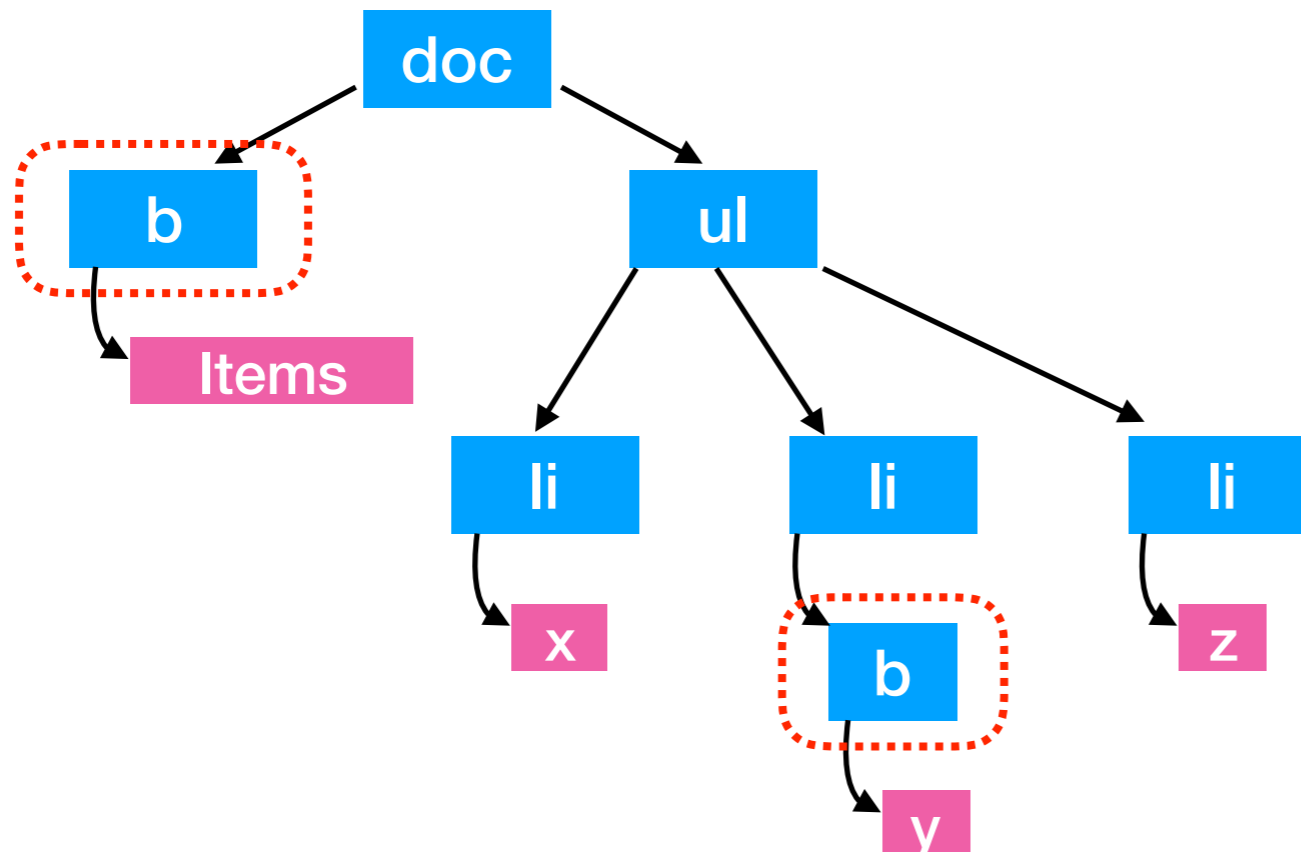
# Searching for Elements

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("b")  
print(len(elements))
```

now look for all bold elements

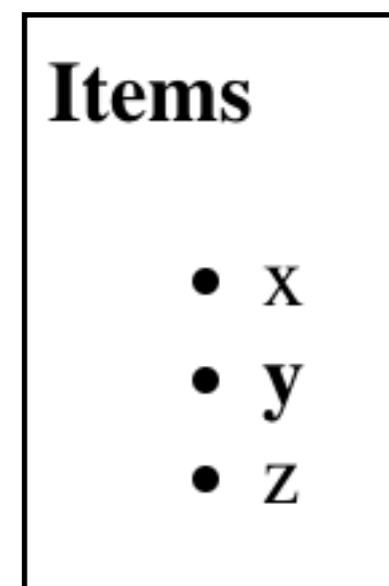
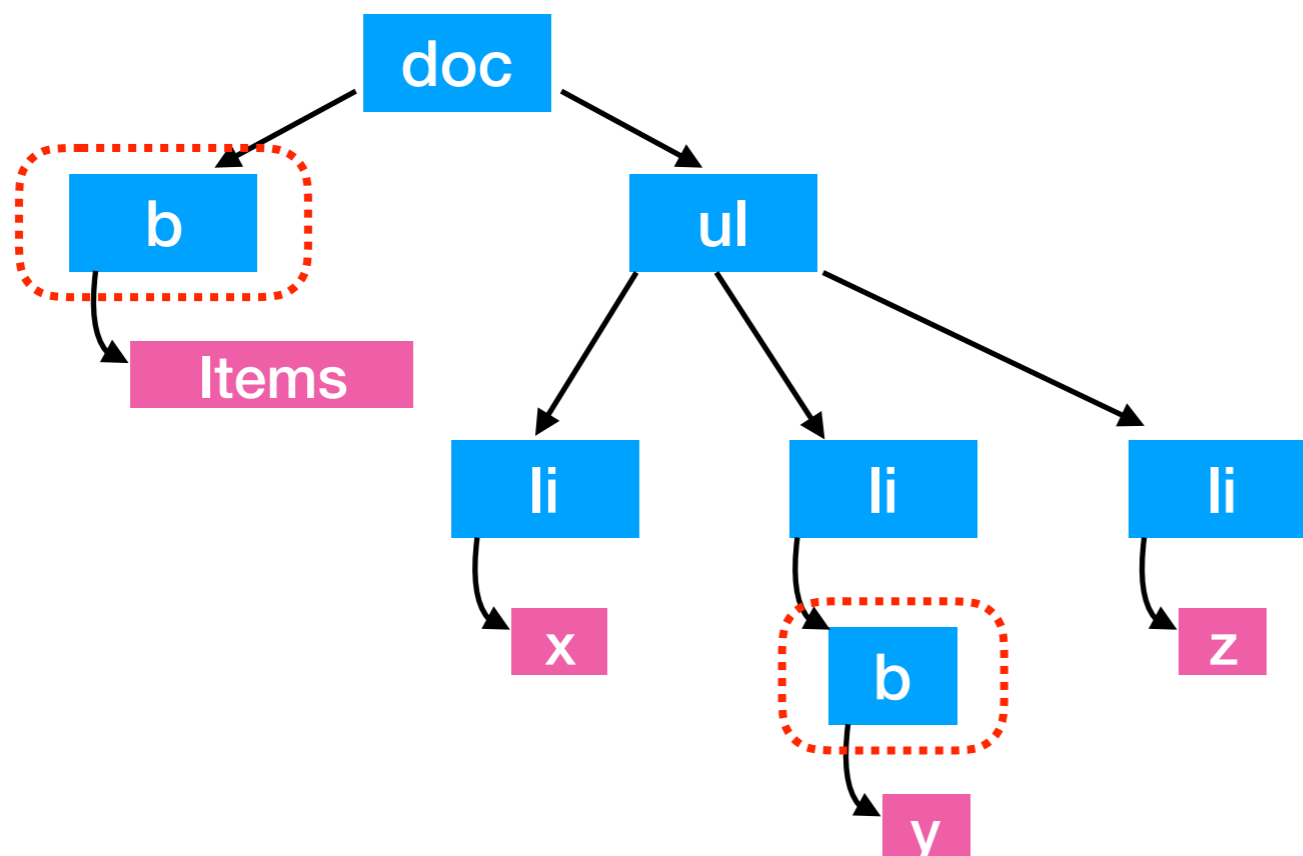


# Searching for Elements

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
elements = doc.find_all("b") list of two elements  
print(len(elements)) prints 2
```



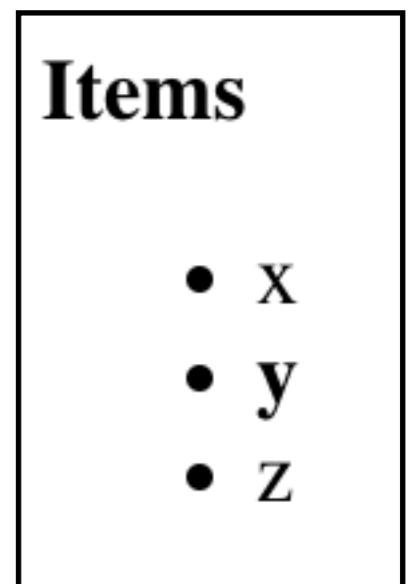
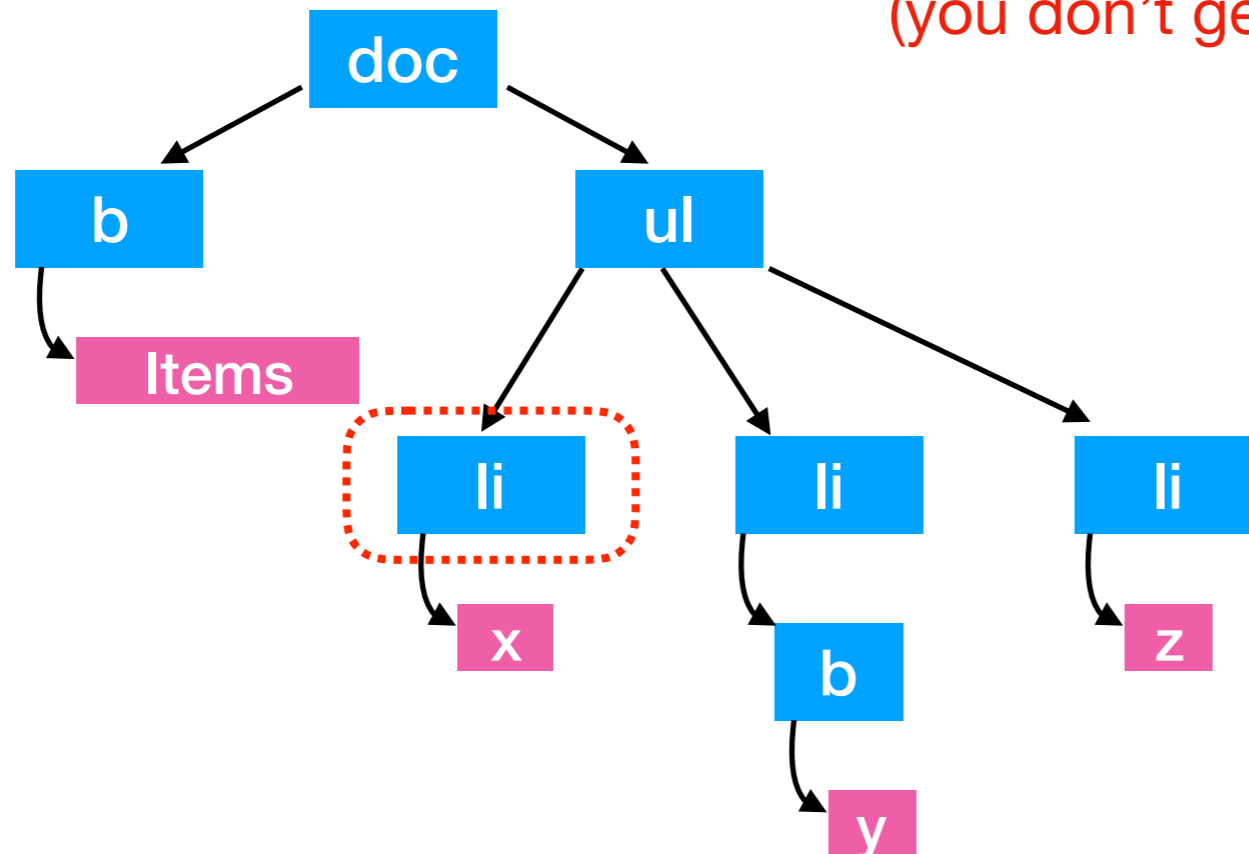
# Find One

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
li = doc.find("li")  
assert(li != None)
```

find just grabs the first one  
(you don't get a list)



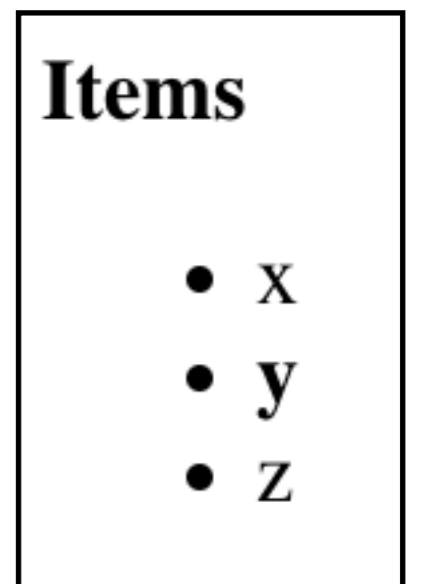
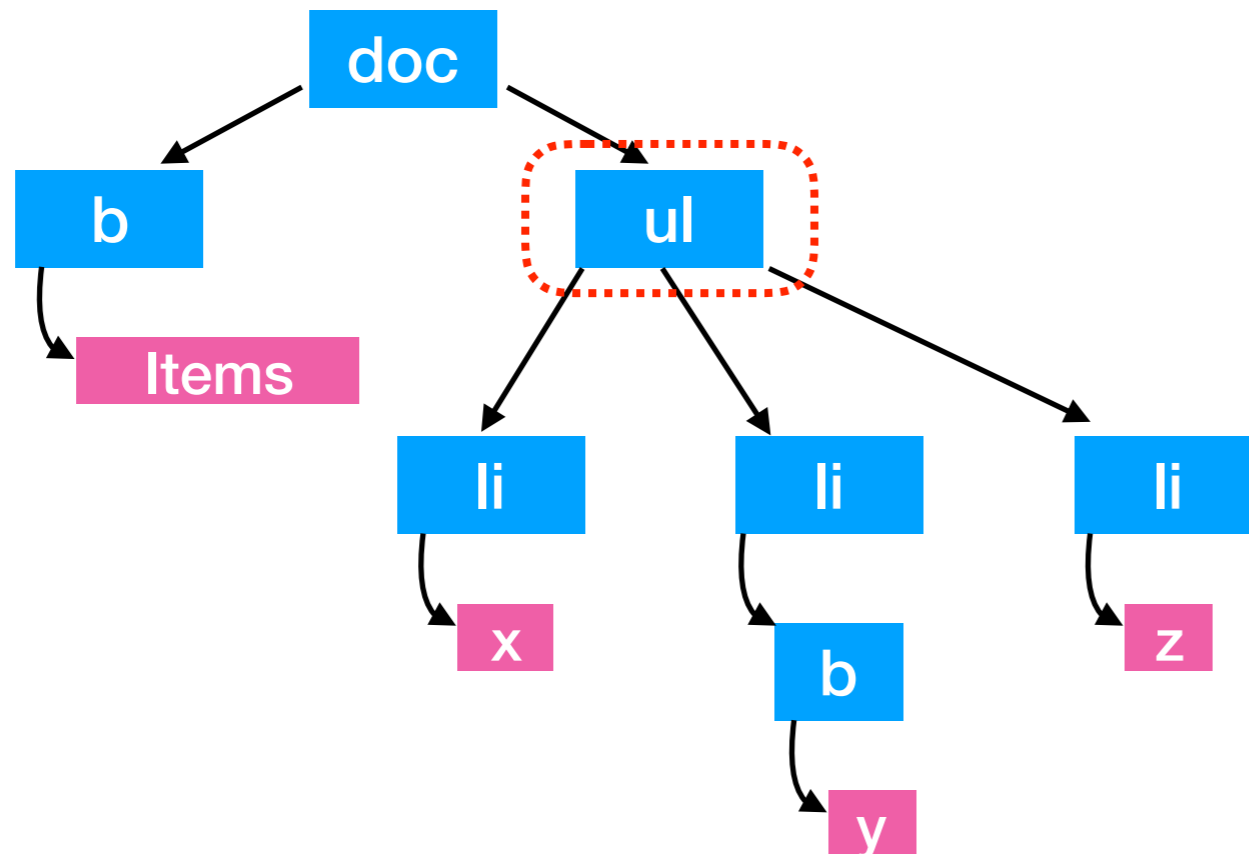


# Find One

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
ul = doc.find("ul")  
assert(ul != None)
```



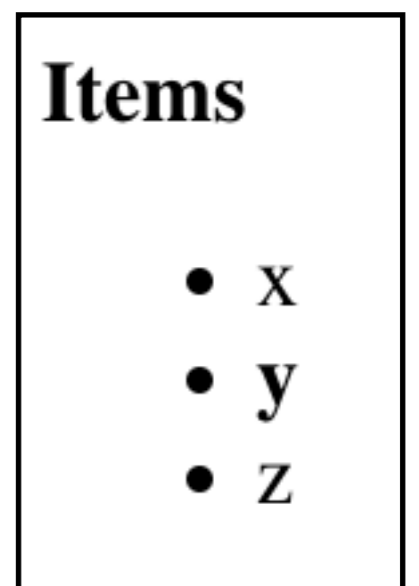
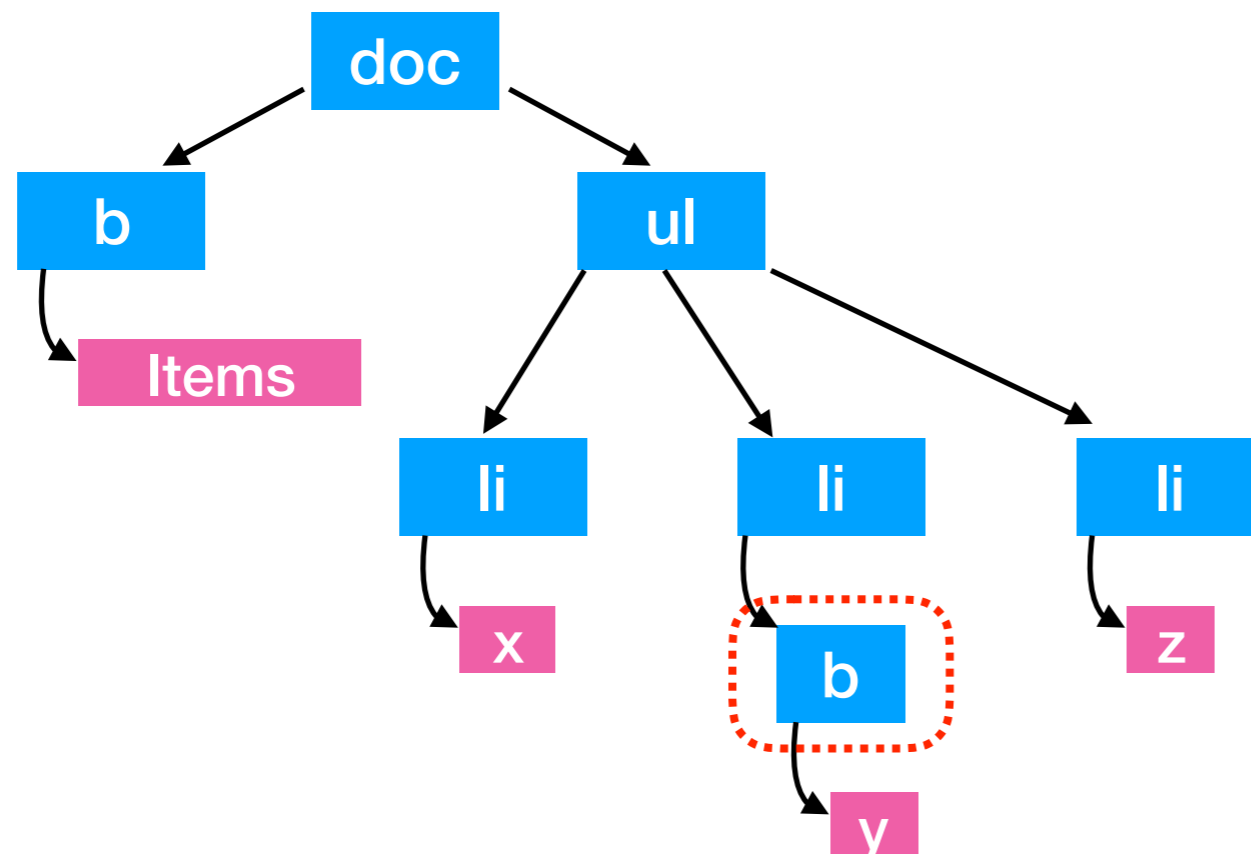
# Search Within Search Results

```
from bs4 import BeautifulSoup
```

```
html = "<b>Items</b><ul><li>x</li><li><b>y</b></li><li>z</li></ul>"  
doc = BeautifulSoup(html, "html.parser")
```

```
ul = doc.find("ul")
```

```
ul.find_all("b") find all bold text in the unordered list
```



# Inspecting an Element

Remember! **Elements** may contain:

- attributes
- text
- other **elements**

# Inspecting an Element

Remember! **Elements** may contain:

- attributes
- text
- other **elements**

---

[what you see]

[please click here](#)

---

[HTML]

```
<a href="schedule.html"><i>please</i> click <b>here</b></a>
```

---

```
link = doc.find("a")
```

[Python]

# Inspecting an Element

Remember! **Elements** may contain:

- attributes
- text
- other **elements**

---

[what you see]

[\*please\* click \*\*here\*\*](#)

---

[HTML]

```
<a href="schedule.html"><i>please</i> click <b>here</b></a>
```

---

[Python]

```
link = doc.find("a")
list(link.children)
```

**Result:**

italic element	click text	bold element
----------------	------------	--------------

  
(list)

# Inspecting an Element

Remember! **Elements** may contain:

- attributes
- text
- other **elements**

---

[what you see]

[please click here](#)

---

[HTML]

```
<a href="schedule.html"><i>please</i> click <b>here</b></a>
```

---

[Python]

```
link = doc.find("a")
link.get_text()
```

**Result:** please click here  
(str)

# Inspecting an Element

Remember! **Elements** may contain:

- attributes
- text
- other **elements**

---

[what you see]

[please click here](#)

---

[HTML]

```
<a href="schedule.html"><i>please</i> click <b>here</b></a>
```

---

[Python]

```
link = doc.find("a")  
link.attrs
```

```
Result: {'href': 'schedule.html'}  
          (dict)
```

# Outline

Document Object Model

BeautifulSoup module

Scraping States from Wikipedia



# Demo Stage 1: Extract Links from Wikipedia

Goal: scrape links to all articles about US states from a table on a wiki page (check this: <https://simple.wikipedia.org/robots.txt>)

## Input:

- [https://simple.wikipedia.org/wiki/List\\_of\\_U.S.\\_states](https://simple.wikipedia.org/wiki/List_of_U.S._states)

## Output:

- <https://simple.wikipedia.org/wiki/Alabama>
- <https://simple.wikipedia.org/wiki/Alaska>
- etc

### List of U.S. states

---

From Wikipedia, the free encyclopedia

A **U.S. state** is one of the [states](#) of the [United States of America](#). Four states (Kentucky, Massachusetts, Pennsylvania, and the twenty-first, 1959.

The states are labeled with their [U.S. postal abbreviations](#), their founding date and [capitals](#).

Sl no. ↕	Abbreviations ↕	State Name ↕	Capital ↕	Became a State ↕
1	AL	Alabama	Montgomery	December 14, 1819
2	AK	Alaska	Juneau	January 3, 1959
3	AZ	Arizona	Phoenix	February 14, 1912
4	AR	Arkansas	Little Rock	June 15, 1836

# Demo Stage 2: Download State Pages

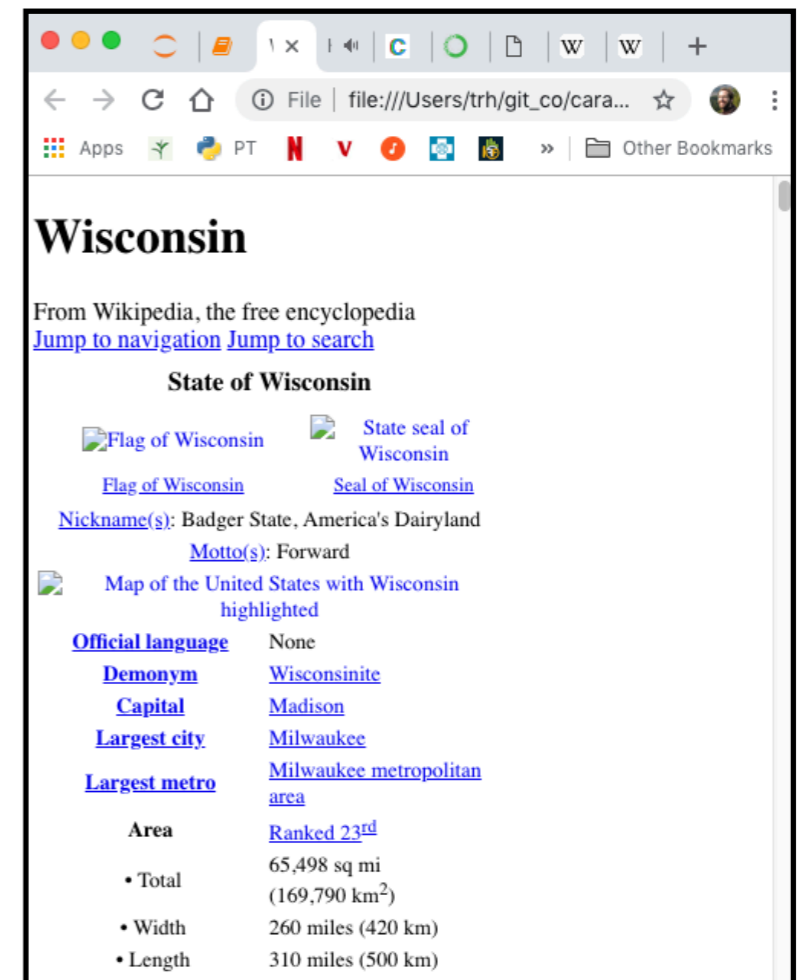
Goal: download all Wiki pages for the states

## Input:

- Links generated in stage 1:
- <https://simple.wikipedia.org/wiki/Alabama>
- <https://simple.wikipedia.org/wiki/Alaska>
- etc

## Output Files:

- Alabama.html
- Alaska.html
- etc



# Demo Stage 3: Convert to DataFrame

state	Abbreviations	Admission to Union	Area	Area-% water	Area-Latitude	Area-Length	Area-Longitude	Area-Total	Area-Width	...	east of 169° 30'	eastern half	most of state	pri
Ohio	OH[14]	March 1, 1803[12] (17th, declared retroactively...	Ranked 34th	8.7	38°24' N to 41° 59' N	220 miles (355 km)	80°31' W to 84°49' W	44,825 sq mi (116,096 km2)	220 miles (355 km)	...	NaN	NaN	NaN	
North_Carolina	NC, N.C.	November 21, 1789 (12th)	Ranked 28th	9.5	33°50' N to 36° 35' N	560[5] miles (901 km)	75°28' W to 84°19' W	53,819 sq mi (139,390 km2)	186 miles (272 km)	...	NaN	NaN	NaN	
Oregon	OR, Ore.	February 14, 1859 (33rd)	Ranked 9th	2.4	42° N to 46°18' N	360 miles (580 km)	116°28' W to 124°38' W	98,381 sq mi (254,806 km2)	400 miles (640 km)	...	NaN	NaN	NaN	Pa
Louisiana	LA, La.	April 30, 1812 (18th)	Ranked 31st	15	28°56' N to 33° 01' N	379 miles (610 km)	88°49' W to 94°03' W	52,378.13 sq mi (135,382 km2)	130 miles (210 km)	...	NaN	NaN	NaN	
Illinois	IL, Ill.	December 3, 1818 (21st)	Ranked 25th	3.99	36°58' N to 42° 30' N	390 miles (628 km)	87°30' W to 91°31' W	57,914 sq mi (149,997 km2)	210 miles (338 km)	...	NaN	NaN	NaN	