MSc in Finance Pre-Term Course (Week5):

Introduction to Web Scraping with Python

Tutor: Yuan Lu

CUHK Department of Finance

yuan.lu@link.cuhk.edu.hk

Aug 15, 2023

In this week, students will learn how to fetch web pages and parse useful information from HTML code.

To accomplish this, the class will cover the **requests** and **BeautifulSoup** libraries, and employ the **pandas** library to manipulate the scraped data.

Additionally, if time permits, the class will give a brief introduction to the **selenium** package for interacting with Javascript-oriented sites, as well as the **scrapy** package for recursively crawling multiple web pages.

Agenda

1. Simple Scrapers for Beginners
    a. Scraping tables with Pandas using read_html()
2. Introduction to HTML
    a. How to understand HTML: Core Elements in HTML
3. Parsing HTML with BeautifulSoup
    a. Getting Elements by Hierarchy
    b. Getting Elements by Search
    c. Several Examples and In-Class Exercises

Section 1:

A Simple Scraper for Beginners

In this section, we learn to use pandas.read_html to extract tables from websites.

#1

Environment Setup

Python Packages and Environments

- Part of Python's appeal is its huge ecosystem of add-ons, or "packages"
- We're going to use several Python packages to help us in our scraping
    - Like *Pandas*, which we've installed it already
    - Also quite a few new ones, such as *requests* and *beautifulsoup* (Other packages such as *selenium* and *scrapy* might not be covered in this week)

```
pip install requests
pip install beautifulsoup4
```

#2

Scraping tables with Pandas using read_html()

Pandas.read_html()

- It is one of the easiest ways to read HTML tables directly from a webpage into a Pandas DataFrame without knowing how to scrape a website's HTML, this tool can be useful for swiftly combining tables from numerous websites.
- Documentation of pandas.read_html: https://pandas.pydata.org/docs/reference/api/pandas.read_html.html

Example 1: Using an HTML string

- We pass the html_string to pd.read_html function, which will extract all the HTML tables and returns a list of all the tables.
- How to understand the HTML will be discussed in the later section and we will go back to this part then.

Example 1: Using an HTML string (Sample Code)

```
import pandas as pd

html_string = '''
<table>
<tr>
    <th>Company</th>
    <th>Country</th>
</tr>
<tr>
    <td>Apple Inc.</td>
    <td>United States</td>
</tr>
<tr>
    <td>Tencent Holdings Limited</td>
    <td>China</td>
</tr>
</table>
'''

df_1 = pd.read_html(html_string)
df_1
```

```
[                   Company        Country
 0                 Apple Inc.  United States
 1  Tencent Holdings Limited          China]
```

Example 2: Reading HTML Data From URL

- We want to extract the List of S&P 500 component stocks from the wikipedia website: https://en.wikipedia.org/wiki/List_of_S%26P_500_companies.
- In case students cannot get access to wikipedia, please try the alternative website: https://www.thelists.org/sp-500-list.html. Though the website didn't provide the complete list, we just use it to try the functionality of read_html().

```
import pandas as pd
import numpy as np

dfs = pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')
len(dfs)
```

2

Example 2: Reading HTML Data From URL (Cont.)

```
# If you check out the Wikipedia List of S&P500 Companies,
# you'll notice there is a table containing the current S&P500 components and a table listing the historical changes
# Let's grab the first table.
dfs[0]
```

```
    Symbol    Security    GICS Sector    GICS Sub-Industry    Headquarters Location    Date added    CIK    Founded
0    MMM    3M    Industrials    Industrial Conglomerates    Saint Paul, Minnesota    1957-03-04    66740    1902
1    AOS    A. O. Smith    Industrials    Building Products    Milwaukee, Wisconsin    2017-07-26    91142    1916
2    ABT    Abbott    Health Care    Health Care Equipment    North Chicago, Illinois    1957-03-04    1800    1888
3    ABBV    AbbVie    Health Care    Pharmaceuticals    North Chicago, Illinois    2012-12-31    1551152    2013 (1888)
4    ACN    Accenture    Information Technology    IT Consulting & Other Services    Dublin, Ireland    2011-07-06    14673
...    ...    ...    ...    ...    ...    ...    ...    ...
498    YUM    Yum! Brands    Consumer Discretionary    Restaurants    Louisville, Kentucky    1997-10-06    1041061    1997
499    ZBRA    Zebra Technologies    Information Technology    Electronic Equipment & Instruments    Lincolnshire, Illinois
500    ZBH    Zimmer Biomet    Health Care    Health Care Equipment    Warsaw, Indiana    2001-08-07    1136869    1927
501    ZION    Zions Bancorporation    Financials    Regional Banks    Salt Lake City, Utah    2001-06-22    109380    1873
502    ZTS    Zoetis    Health Care    Pharmaceuticals    Parsippany, New Jersey    2013-06-21    1555280    1952
503 rows * 8 columns
```

Example 2: Reading HTML Data From URL (Cont.)

- Alternative website: https://www.thelists.org/sp-500-list.html

```
import pandas as pd
import numpy as np

dfs1 = pd.read_html('https://www.thelists.org/sp-500-list.html')
dfs1[0]
```

```
    S&P 500 List
0    3M Company
1    Abbott Laboratories
2    Abercrombie & Fitch Co.
3    Adobe Systems
4    Advanced Micro Devices
...     ...
453    XTO Energy Inc.
454    Yahoo Inc.
455    Yum! Brands Inc
456    Zimmer Holdings
457    Zions Bancorp
458 rows * 1 columns
```

Section 2:

HTML

In this section, we will have an overview of HTML, learn the core elements in HTML so as to be able to read HTML.

1. know the differences between common HTML elements: `div`, `p`, `a`, `table`, `span`

1. use the browser to determine what HTML element contains a given piece of data.

2. uniquely describe the HTML element containing a certain piece of data using its type, class, and/or ID.

#1

Overview of HTML

HyperText Markup Language

- HTML is a way of *encoding information* that describes what to display in a web browser
- Every page you see while surfing the web is just a huge document of HTML code
- Why do we need to know it?
  - Scraping sites really boils down to fetching the right HTML and extracting information from it
  - To do that, you need to be able to understand HTML at a basic level
  - **A good web scraper should be able to *read* HTML, but not necessarily *write* it**

HyperText Markup Language(Cont.)

- HTML uses **text-based "tags"** to denote elements in a page
  - e.g. The `<p>` tag represents a paragraph of text, the `<img>` tag represents an embedded image
- Tags are surrounded by chevrons to set them apart from regular text on the page
- `<h1>`, `<p>`, `<br>`, `<div>`

HTML Enclosing Tags

- Some tags **enclose**, or contain, other content. That content can be simple text or more HTML code.
  - Paragraph tags enclose the text of the paragraph
  - Heading tags enclose the text of the heading
  - Divider tags contain other HMTL that lives within that division of the page

HTML Enclosing Tags

- Some tags **enclose**, or contain, other content. That content can be simple text or more HTML code.
  - Paragraph tags enclose the text of the paragraph
  - Heading tags enclose the text of the heading
  - Divider tags contain other HMTL that lives within that division of the page
- Enclosing tags have two parts: an **opening tag** (which comes before the enclosed content) and a **closing tag** (which comes after)

```
<h1>My Heading</h1>
<p>And some more text I wrote</p>
```

- Note that the closing tag has a forward slash (/) before the tag name to distinguish it from an opening tag.

HTML Empty Tags

- The opposite of an enclosing tag is an **empty tag**, which doesn't contain anything else inside it.
  - Image tags, line break tags

HTML Empty Tags

- The opposite of an enclosing tag is an **empty tag**, which doesn't contain anything else inside it.
    - Image tags, line break tags

These tags have only one part, no opening or closing.

```
Some text before a line break...
<br>
...and some more after.
```

```
<img src="assets/my_photo.png">
```

- You will sometimes see empty tags represented with a foward slash before the second chevron (`<br/>`) but generally this isn't necessary.

HTML Examples

```
<h1>Welcome</h1>

<p>This is our fifth class. <br> Hope you enjoy our class!</p>
```

HTML Examples

```
<h1>Welcome</h1>

<p>This is our fifth class. <br> Hope you enjoy our class!</p>
```

Welcome

This is our fifth class.
Hope you enjoy our class!

HTML Examples (Cont.)

```
<h1>Pre-Term Course</h1>

<h2>Week-5</h2>
<p>Introduction to HTML</p>

<h2>Week-6</h2>
<p>Final Exam.</p>
```

HTML Examples (Cont.)

```
<h1>Pre-Term Course</h1>

<h2>Week-5</h2>
<p>Introduction to HTML</p>

<h2>Week-6</h2>
<p>Final Exam.</p>
```

Pre-Term Course

Week-5

Introduction to HTML

Week-6

Final Exam.

HTML Examples (Cont.)

```
<p>Click <a href="https://google.com">here</a> to go to google.com</p>
```

HTML Examples (Cont.)

```
<p>Click <a href="https://google.com">here</a> to go to google.com</p>
```

Click here to go to google.com

#2

Core HTML Elements to Know

p Tag

- The **paragraph** tag
- More generally, often used as a container for any text (not just a paragraph)
    - Though it's not *required* -- text can go directly inside many elements

```
<p>Today, we are going to talk about how to read HTML!</p>
```

span Tag

- The **span** tag
- Used to some part of a larger body of text, to style in differently

```
<p>Some text is <span>very special</span> and should be thought of separately from the rest.</p>
```

span Tag

- The **span** tag
- Used to some part of a larger body of text, to style in differently

```
<p>Some text is <span>very special</span> and should be thought of separately from the rest.</p>
```

- e.g. make some words big and red
- We'll talk more about styling later

Heading Tags

- `h1`, `h2`, `h3`, `h4`, `h5`
- These tags are for headings, and automatically make the text larger
  - They usually add some padding around the edges too, to space it away from other elements
- `h1` is a top-level heading, like a page title. `h2` is a little smaller, like a section heading. `h3` is yet smaller, etc.

Heading Tags

- `h1`, `h2`, `h3`, `h4`, `h5`
- These tags are for headings, and automatically make the text larger
    - They usually add some padding around the edges too, to space it away from other elements
- `h1` is a top-level heading, like a page title. `h2` is a little smaller, like a section heading. `h3` is yet smaller, etc.

```
<h1>Main Title</h1>
<h3>Subtitle</h3>
<h5>Mini-heading</h5>
```

Main Title

Subtitle

Mini-heading

img Tag

- The **image** tag
- Represents a picture to be embedded in the page.
  - The image itself is usually stored on the internet somewhere (maybe in another place on the same website)
  - The image tag uses a link to that location as its "source"

img Tag

- The **image** tag
- Represents a picture to be embedded in the page.
  - The image itself is usually stored on the internet somewhere (maybe in another place on the same website)
  - The image tag uses a link to that location as its "source"

```
<img src="/assets/panda.jpg"/>
```

a Tag

- The **anchor** tag, is used for **links** to other pages
  - The tag encloses the text that will become a link

a Tag

- The **anchor** tag, is used for **links** to other pages
  - The tag encloses the text that will become a link

```
<p>Click <a href="https://en.ipanda.com/">here</a> to see the ipanda website.</p>
```

Click here to see the ipanda website.

a Tag

- The **anchor** tag, is used for **links** to other pages
  - The tag encloses the text that will become a link

```
<p>Click <a href="https://en.ipanda.com/">here</a> to see the ipanda website.</p>
```

Click here to see the ipanda website.

- Note that this tag uses an additional *attribute*, href.
  - href means HyperText Reference, and it must be set to the page where the link should lead

div Tag

- The **Content Division** element
- Arguably (in my mind) the cornerstone of HTML
- This is, by and large, how other elements are grouped together

```
<div>
  <h2>Chapter 1</h2>
  <img src="/images/picture1.png"/>
  <p>What is HTML?</p>
</div>
<div>
  <h2>Chapter 2</h2>
  <img src="/images/picture2.png"/>
  <p>How to read HTML?</p>
</div>
```

div Tag

- The **Content Division** element
- Arguably (in my mind) the cornerstone of HTML
- This is, by and large, how other elements are grouped together

```
<div>
  <h2>Chapter 1</h2>
  <img src="/images/picture1.png"/>
  <p>What is HTML?</p>
</div>
<div>
  <h2>Chapter 2</h2>
  <img src="/images/picture2.png"/>
  <p>How to read HTML?</p>
</div>
```

- Note how we indent HTML code as it becomes nested.

`table` Tag

- The **table** tag
- Maybe not as foundational as the above tags, but important to us for tabular data
- Tables are sometimes used to display data, but also sometimes used to lay out parts of a page
  - e.g. breaking text into columns, sidebars, etc.
- Relatively complicated HTML structure in order to denote rows & columns.

`table` Tag (Cont.)

```
<table>
<tr>
    <th>Company</th>
    <th>Country</th>
</tr>
<tr>
    <td>Apple Inc.</td>
    <td>United States</td>
</tr>
<tr>
    <td>Tencent Holdings Limited</td>
    <td>China</td>
</tr>
</table>
```

Company	Country
Apple Inc.	United States
Tencent Holdings Limited	China

- Looking for HTML tables is a big part of `pd.read_html`

Meta Elements

- There are some "meta" elements to know as well
  - `title` -- The text enclosed by this tag becomes the title of the page (shows in your browser tab)
  - `meta` -- Defines metadata about an HTML document. Metadata is data (information) about data, which will not be displayed on the page, but is machine parsable.
  - `script` -- Encloses or links to other code used by the page, often JavaScript
  - `head` -- The element is a container for metadata (data about data) and is placed between the `<html>` tag and the `<body>` tag
  - `body` -- The opposite of `head`; the elements that make up the visible part of the page are usually in here
  - `html` -- The entire content of a page is typically contained within `<html>...</html>` to indicate it's HTML code.

Other Elements

- There are many, many other elements
- But remember, we're here to read HTML, not write it ourselves
    - If you know the general structure of elements, you can figure most of them out if you run across them

#3

Looking at HTML in the Browser

HTML in the Browser

- Most (maybe all) browsers have a set of *Developer Tools*, which allow you to inspect various things about web pages
- These tools are invaluable when scraping the web; they help you track down elements that contain the info you want
- Right click on some text on the page and select *Inspect Element* or *Inspect*
    - This should be available out-of-the-box in Chrome and Firefox
    - In Safari, you'll have to enable it: go to Preferences > Advanced > Show develop menu in menu bar

Demo

Inspect (zssh000001) guba.eastmoney

https://guba.eastmoney.com/list,zssh000001.html

#4

Tag Attributes

Attribute Example

- In looking at HTML in the browser, we saw lots of tags that were more complicated than the simple ones we learned in previous part

```
<div id="nav-bar" class="nav">
  ...
</div>
```

- Here, we would say that the div has two **attributes**: `id` and `class`
    - These are two of the most common attributes of HTML tags, though there are many more

Attribute Example

- In looking at HTML in the browser, we saw lots of tags that were more complicated than the simple ones we learned in previous part

```
<div id="nav-bar" class="nav">
  ...
</div>
```

- Here, we would say that the div has two **attributes**: `id` and `class`
  - These are two of the most common attributes of HTML tags, though there are many more
- Each attribute also has a **value**
  - The value of the id attribute is "nav-bar"
  - The value of the class attribute is "nav"

Attribute Example (Cont.)

- We've already seen a couple of attributes

Attribute Example (Cont.)

- We've already seen a couple of attributes
- Images have a `src` attribute to point to the image file

```
<img src="assets/image.png"/>
```

Attribute Example (Cont.)

- We've already seen a couple of attributes
- Images have a `src` attribute to point to the image file

```
<img src="assets/image.png"/>
```

- Links (anchor tags) have an `href` attribute to hold the linked URL

```
<a href="https://google.com">Go to Google</a>
```

Types of Attributes

- `src` and `href` are examples of attributes that only make sense on certain elements
  - It doesn't make much sense to have a "source" for a paragraph, or a HyperText Reference for a content division
- But certain tags are applicable to pretty much all elements, most prominently:
  - **id**
  - **class**
  - **style**

ID Attribute

- A unique name (within the current page) to identify the element

```
<img src="/assets/logo.png" id="site-logo"/>
```

- There must only one element on the page with this ID.

Class Attribute

```
<h2 class="chapter-title">Chapter 7: Salamander</h2>
<p>
  ...
</p>
<h2 class="chapter-title">Chapter 8: Rat</h2>
<p>
  ...
</p>
```

- Both chapter titles are the same type of thing
  - Probably are the same font, size, color, etc
  - Marking them as the same class makes it easier to standardize them

Style Attribute

- The style attribute allows styling (think color, size, font type, padding) to be applied to an element

```
<img src="/assets/logo.png" style="width:100%;padding:3px;"/>
```

- *But* styling of elements is more traditionally done in separate files, so you won't see this too much

HTML Styling & CSS (Skip)

- HTML styling is another big topic
  - We will not cover it in our class
- Good to know a few things though:
  - Styles are usually applied through separate files called Cascading Stylesheets (CSS), or sometimes just "stylesheets"
  - Styles are applied to given elements based on type (e.g. `div`), ID, and class
  - That's the point of the ID and class attributes, in case you were wondering -- to mark elements for certain styling

A Quick Peek at CSS (Skip)

```
/* Make chapter titles big and red */
h2.chapter-title {
  color: red;
  font-size: 3em;
}

/* Add some padding around our site logo image */
#site-logo {
  padding: 20px;
}
```

Section 3:

Parsing HTML with BeautifulSoup

Now that we know a bit about HTML, we can use the BeautifulSoup package to extract information from it.

1. Use Python to programmatically fetch an arbitrary bit of data on a static page via BeautifulSoup.

#1

BeautifulSoup Overview

What's BeautifulSoup?

- Direct from the documentation:

  Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

- **"pulling data out of HTML and XML files"** -- Extracting data that is stored as part of a web page, programmatically

What's BeautifulSoup?

- Direct from the documentation:

  Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

- **"pulling data out of HTML and XML files"** -- Extracting data that is stored as part of a web page, programmatically

- **favorite parser** -- BeautifulSoup can use several other packages as its "parser", a tool to break down the HTML code. For simplicity, we'll use a parser that's built into Python: `html.parser`

What's BeautifulSoup?

- Direct from the documentation:

  Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

- **"pulling data out of HTML and XML files"** -- Extracting data that is stored as part of a web page, programmatically

- **favorite parser** -- BeautifulSoup can use several other packages as its "parser", a tool to break down the HTML code. For simplicity, we'll use a parser that's built into Python: `html.parser`

- **idiomatic ways of navigating, searching, and modifying the parse tree** -- The best part of BeautifulSoup isn't that it can break down HTML; it's that it provides a very intuitive interface for doing that

What's BeautifulSoup?

- Direct from the documentation:

  Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

- **"pulling data out of HTML and XML files"** -- Extracting data that is stored as part of a web page, programmatically
- **favorite parser** -- BeautifulSoup can use several other packages as its "parser", a tool to break down the HTML code. For simplicity, we'll use a parser that's built into Python: `html.parser`
- **idiomatic ways of navigating, searching, and modifying the parse tree** -- The best part of BeautifulSoup isn't that it can break down HTML; it's that it provides a very intuitive interface for doing that
- Ultimately, BeautifulSoup is the standard tool for **Post-processing** raw HTML code

Interface

- Everything starts with a `BeautifulSoup` object, which takes the HTML code as an argument in order to parse it.
    - Also takes the name of the parser we'll be using, which will be `'html.parser'` for the duration of this class

```
from bs4 import BeautifulSoup

html = '<h1>Welcome</h1><p>This is my website.</p>'
bs = BeautifulSoup(html)
```

Interface

- Printing the soup object shows you the HTML that it wraps

```
print(bs)
```

```
<html><h1>Welcome</h1><p>This is my website.</p></html>
```

- Notice how it added `html` tags

Interface

- Printing the soup object shows you the HTML that it wraps

```
print(bs)
```

```
<html><h1>Welcome</h1><p>This is my website.</p></html>
```

- Notice how it added `html` tags
- It's much easier to read HTML that's been indented properly, so there is also a `prettify()` method.

```
print(bs.prettify())
```

```
<html>
<body>
    <h1>
        Welcome
    </h1>
    <p>
        This is my website.
    </p>
</body>
</html>
```

Interface

- The really good parts of the interface are for navigating and searching for elements in the HTML "tree", which we'll talk about next.

#2

Getting Elements by Hierarchy

The HTML Tree

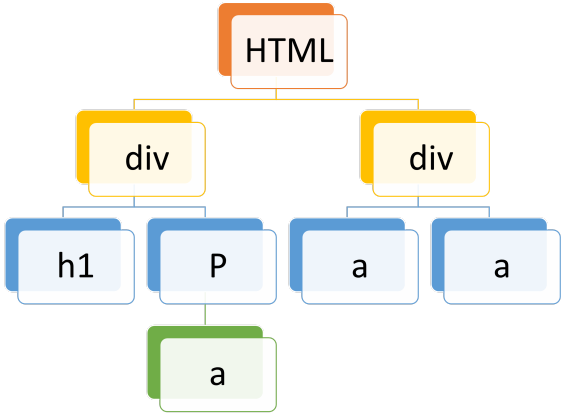
- Much of BeautifulSoup's power comes from navigating through HTML
- In order to make use of that, we need to think about HTML the way it does -- as a tree

The HTML Tree

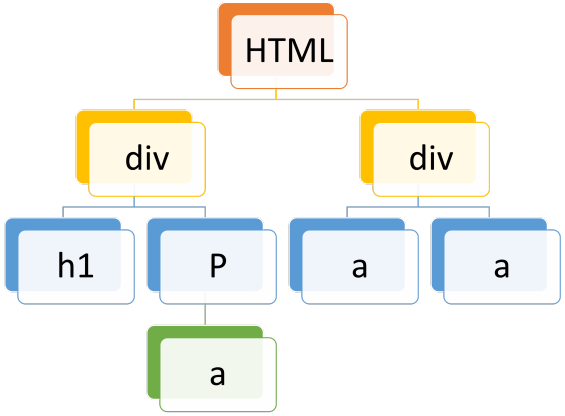
```
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
```

The HTML Tree

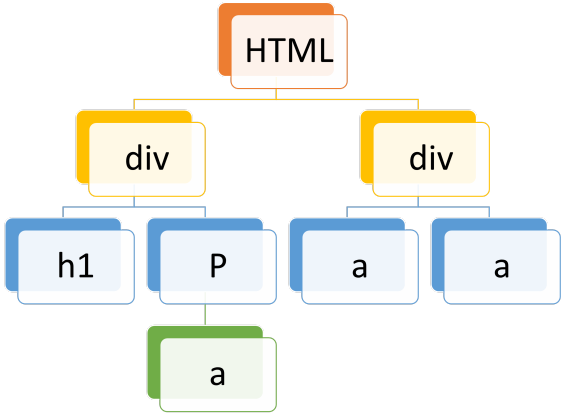
```
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
```



The HTML Tree

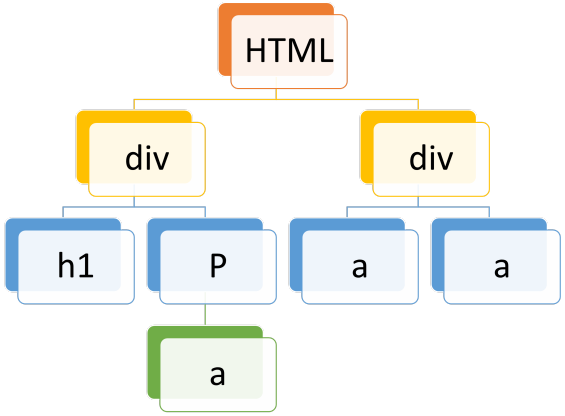


The HTML Tree



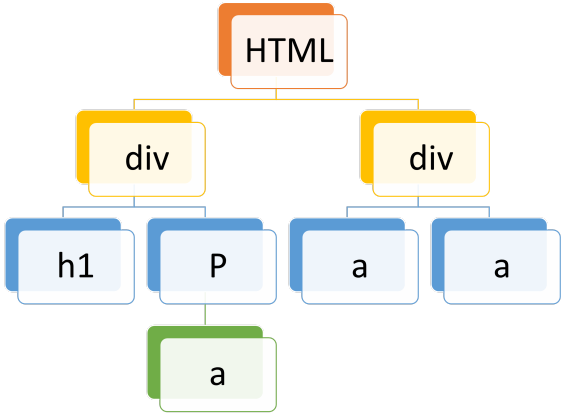
- A **child** is an element that lives within another element, and appears below it on the tree, connected by a black line
    - Both divs are children of the `<html>` tag, the leftmost `<a>` tag is a child of the `<p>` tag

The HTML Tree



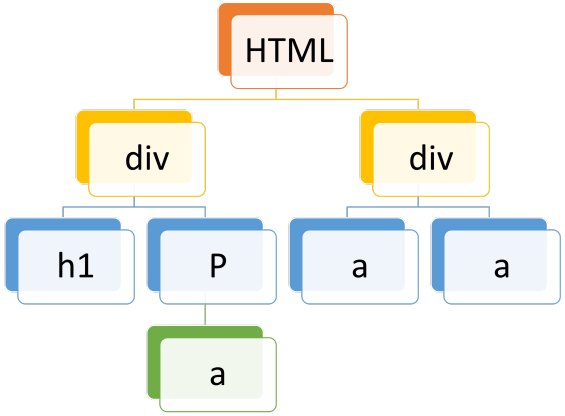
- A **parent** is an element that contains another element -- the exact inverse of a child
  - The HTML element is the parent of both divs, the `<p>` tag is the parent of the leftmost `<a>` tag

The HTML Tree



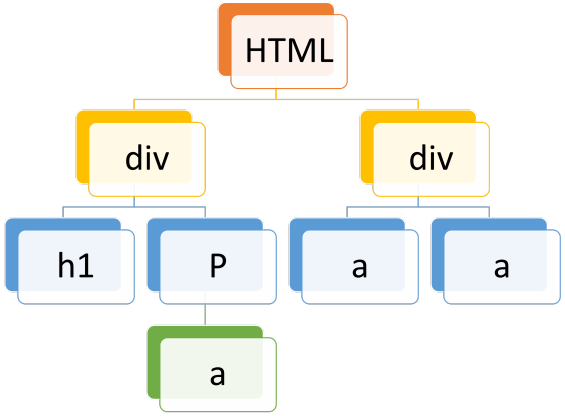
- A parent can have any number of child elements (sometimes lots!), but every child has exactly one parent element.
    - The only element without a parent is the HTML tag, which we thus call the "root" of the tree

The HTML Tree



- Child elements that share the same parent are called **siblings**
  - The two divs are siblings, as are the `<h1>` and `<p>` tags

The HTML Tree



- How would we describe the relationship between the `<h1>` and `<html>` tags? They're not directly connected.
    - We sometimes say "descendents" to mean all children-of-children, and "ancestors" to mean all parents-of-parents

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.html.div.h1
```

```
<h1>Shenzhen Stock Exchange</h1>
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.h1
```

```
<h1>Shenzhen Stock Exchange</h1>
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.p
```

```
<p>SSE Composite Index is composed of all eligible stocks and CDRs listed on Shanghai Stock Exchange.</p>
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.p.a
```

```
<a href="http://www.szse.cn/English/index.html">Home page</a>
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.p.a.string
```

```
'Home page'
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.a
```

```
<a href="http://www.szse.cn/English/index.html">Home page</a>
```

BS Dot Syntax

- Using dots (`bs.div.p`), you can navigate "down" the tree
- BeautifulSoup will find the *first element* of that type
    - It will look at both children and further descendents (children of children, ... of children)

BS Dot Syntax

- How can we get the link to "Stocks"?

```
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
```

BS Dot Syntax

- How can we get the link to "Stocks"?

```
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
```

- Sometimes simply finding the first instance of an element isn't enough

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.next_sibling.next_sibling.a
```

```
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.next_sibling.next_sibling.a
```

```
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
```

- We move over by *two* siblings instead of one because the whitespace between the divs is itself a sibling.

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.h1.next_sibling
```

```
'\n'
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.h1.next_sibling.next_sibling
```

```
<p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the supervision o
 <a href="http://www.szse.cn/English/index.html">Home page</a></p>
```

BS Dot Syntax

- You can also use `.previous_sibling`, which does exactly what it sounds like
  - I find it less handy though
- Also `.parent` gets you the direct parent of the current element

BS Dot Syntax

- Last, and maybe most importantly, you can get the text inside elements using `.string`
    - **But** this only works on elements that don't contain other things. They need to have only text, or be the parent of an element that contains only text.

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.h1.string
```

```
'Shenzhen Stock Exchange'
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.next_sibling.next_sibling.a.string
```

```
'Stocks'
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.div.p.string
```

(yields `None`)

BS Dot Syntax

- Sometimes you want *all* the text within an element and its children
- You can use `.strings` (note the final `s`) to get a list of all of them.

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
list(bs.div.p.strings)
```

```
['Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the supervision of
 'Home page']
```

BS Dot Syntax

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
list(bs.div.next_sibling.next_sibling.strings)
```

```
['\n', 'Stocks', '\n', 'Funds', '\n']
```

#3

Searching

Searching for Elements

- Navigating the tree is powerful but can get cumbersome in large, deeply-nested HTML
- **Searching** allows us to find elements by their type *or attributes*, including ID and class

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

Previously, to get the marketdata div:

```
bs.div.next_sibling.next_sibling
```

```
<div class="marketdata">
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
</div>
```

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

With searching:

```
bs.find(name='div', class_='marketdata')
```

```
<div class="marketdata">
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
</div>
```

Searching for Elements

- The key to searching is the `.find` method of BeautifulSoup elements
  - Accepts arguments including `name` (the type of tag), `id` (the ID of the tag), and `class_` (the class of the tag)
- It returns the **first element that matches** the criteria
  - Even if that element is several layers deep!

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
bs.find(class_='introduction')
```

```
<div class="introduction">
<h1>Shenzhen Stock Exchange</h1>
<p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the supervision o
  <a href="http://www.szse.cn/English/index.html">Home page</a></p>
</div>
```

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

You can even mix regular dot syntax with searching:

```
bs.find(class_='marketdata').a.string
```

```
'Stocks'
```

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

And you're allowed to search for pretty much any attribute...

```
# What's the text of the link http://www.szse.cn/English/index.html?
bs.find(href='http://www.szse.cn/English/index.html').string
```

```
'Home page'
```

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

You can even search for bits of text, and then get their parent element using dot syntax:

```
bs.find(text='Home page').parent
```

```
<a href="http://www.szse.cn/English/index.html">Home page</a>
```

Searching for Elements

- As you can probably see, the ability to find anything in the HTML based on its name and attributes is incredibly powerful
- You can even find **all matching elements** (instead of just the first one), using `.find_all` instead of `.find`
    - This takes all the same arguments and returns a list of matching elements

Searching for Elements

- We now know how to use attributes and hierarchy to find individual elements, and to use `.string` to get the text inside of them
- But what if we wanted to get the attributes of an element we've found?

Searching for Elements

```
bs.find(class_='marketdata').a
```

```
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
```

Searching for Elements

```
bs.find(class_='marketdata').a
```

```
<a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
```

```
bs.find(class_='marketdata').a['href']
```

```
'http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html'
```

Searching for Elements

```
html = '''
<html>
  <div class="introduction">
    <h1>Shenzhen Stock Exchange</h1>
    <p>Shenzhen Stock Exchange (SZSE), established on 1st December, 1990, is a self-regulated legal entity under the
      <a href="http://www.szse.cn/English/index.html">Home page</a></p>
  </div>

  <div class="marketdata">
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/stocks/index.html">Stocks</a>
    <a href="http://www.szse.cn/English/siteMarketData/siteMarketDatas/funds/index.html">Funds</a>
  </div>
</html>
'''
bs = BeautifulSoup(html, 'html.parser')
```

```
# What link does the text "Home page" point to?
bs.find(text='Home page').parent['href']
```

```
'http://www.szse.cn/English/index.html'
```

#4

Examples

Example-1: Scrape the countries of the world

Scrape the countries of the world and the related metrics from the following site:
https://scrapethissite.com/pages/simple/

Store the result in a DataFrame that looks like the following:

name	capital	population	area
Andorra	Andorra la Vella	84000	468.0

....

Example-1: Sample Solution

This website is very scraping-friendly, but we still have to string together a lot of concept we've been practicing in more contained problems:

- Fetching HTML with `requests`
- Parsing it with the BeautifulSoup class
- Locating elements of interest
- Looping over multiple elements
- Creating a DataFrame from scraped elements

Example-1: Sample Solution (Cont.)

- First of all, when we inpect the website:

```
Request URL: https://www.scrapethissite.com/pages/simple/
Request Method: GET
Status Code: 200
```

- We can pull data using requests.get(url)

Example-1: Sample Solution (Cont.)

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

URL = 'https://scrapethissite.com/pages/simple/'

# Get the site HTML.
response = requests.get(URL)
```

Example-1: Sample Solution (Cont.)

- Parse the HTML with BeautifulSoup
- As for finding the elements, the simplest "container" to loop over is the "col-md-4 country" `div` element -- there is one of these for each country, so we can `find_all()` and then extract the information within each.

```
bs = BeautifulSoup(response.content, 'html.parser')
# Find the divs that contain countries.
divs = bs.find_all(name='div', class_='col-md-4 country')
```

Example-1: Sample Solution (Cont.)

- Step by step

```
# see the first div
print(divs[0].prettify())
```

```
<div class="col-md-4 country">
 <h3 class="country-name">
  <i class="flag-icon flag-icon-ad">
  </i>
  Andorra
 </h3>
 <div class="country-info">
  <strong>
   Capital:
  </strong>
  <span class="country-capital">
   Andorra la Vella
  </span>
  <br/>
  <strong>
   Population:
  </strong>
  <span class="country-population">
   84000
  </span>
  <br/>
  <strong>
   Area (km
```

Example-1: Sample Solution (Cont.)

- Step by step

```
list(divs[0].h3.strings)
```

```
['\n', '\n                              Andorra\n                          ']
```

```
# Get County Name
name = ''.join(divs[0].h3.strings).strip()
name
```

```
'Andorra'
```

```
# Get County Capital
capital = divs[0].find(name='span', class_='country-capital').string
capital
```

```
'Andorra la Vella'
```

- So on and so forth

Example-1: Sample Solution (Cont.)

- For loop

```
# For each one, extract name, capital, population, and area store that info in a dictionary and add it to our list o
rows = []
for div in divs:
    # For name, We can't just use div.h3.string because there is not just text within the h3.
    name = ''.join(div.h3.strings) #Alternative: name = list(divs[0].h3.strings)[1]
    # (Optional) Get rid of whitespace around the country name
    name = name.strip()
    # Everything else is simpler; use the span classes and .string.
    # Capital
    capital = div.find(name='span', class_='country-capital').string
    # Population
    population = div.find(name='span', class_='country-population').string
    # Area
    area = div.find(name='span', class_='country-area').string

    # Create a dictionary of this info
    country_dict = {'name': name, 'capital': capital, 'population': population, 'area': area}
    # Add it to our list of rows
    rows.append(country_dict)

# Now just transform our rows into a DataFrame
country_df = pd.DataFrame(rows)
country_df
```

Example-1: Sample Solution (Cont.)

- Output

```
    name    capital    population    area
0    Andorra    Andorra la Vella    84000    468.0
1    United Arab Emirates    Abu Dhabi    4975593    82880.0
2    Afghanistan    Kabul    29121286    647500.0
3    Antigua and Barbuda    St. John's    86754    443.0
4    Anguilla    The Valley    13254    102.0
...    ...    ...    ...    ...
245    Yemen    Sanaa    23495361    527970.0
246    Mayotte    Mamoudzou    159042    374.0
247    South Africa    Pretoria    49000000    1219912.0
248    Zambia    Lusaka    13460305    752614.0
249    Zimbabwe    Harare    11651858    390580.0
250 rows * 4 columns
```

Example-2: Scrape the posts on Eastmoney Guba.com

Scrape SSE index-related posts information from the following site:
https://guba.eastmoney.com/list,zssh000001.html

We find the urls are changing with the page number.

i.e.:

For the first page: `url = 'https://guba.eastmoney.com/list,zssh000001_1.html'`

For the i-th page: `url = f'https://guba.eastmoney.com/list,zssh000001_{i}.html'`

Store the result in a DataFrame that looks like the following:

read	reply	title	author	update
1028	4	post title	username1	08-15 12:05

Example-2: Sample Solution

- First of all, when we inpect the website:

```
Request URL: https://guba.eastmoney.com/list,zssh000001_1.html
Request Method: GET
Status Code: 200 OK
```

- We can pull data using requests.get()

Example-2: Sample Solution (Cont.)

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

i=1
url=f'https://guba.eastmoney.com/list,zssh000001_{i}.html'

# Get the site HTML without headers.
response = requests.get(url)
response
```

```
<Response [403]>
```

Supplement: Status Codes

- Status codes are a feature of HTTP and provide a simple and quick-to-check way of communicating basics about the "status" of a request/response cycle.
    - 100s -- "Informational Responses"
    - 200s -- "Successful Responses"
    - 300s -- "Redirects" (send you to a new page instead of the one you asked for)
    - 400s -- "Client errors" (the sender of the request did something wrong)
    - 500s -- "Server errors" (the server ran into a problem when trying to create and send a response)

Supplement: Status Codes

- Status codes are a feature of HTTP and provide a simple and quick-to-check way of communicating basics about the "status" of a request/response cycle.
    - 100s -- "Informational Responses"
    - 200s -- "Successful Responses"
    - 300s -- "Redirects" (send you to a new page instead of the one you asked for)
    - 400s -- "Client errors" (the sender of the request did something wrong)
    - 500s -- "Server errors" (the server ran into a problem when trying to create and send a response)
- There are many complete lists of all standard codes, see here for one from Mozilla.

Supplement: Headers

- Headers can make a big difference in the response you get to your request
    - So far we've been assuming that for any URL, a GET request always returns the same thing. But not so!
- Sometimes we can request the response be in a specific format -- HTML or JSON. Getting the best format to parse can make our job easier

Supplement: Headers

- Headers can make a big difference in the response you get to your request
    - So far we've been assuming that for any URL, a GET request always returns the same thing. But not so!
- Sometimes we can request the response be in a specific format -- HTML or JSON. Getting the best format to parse can make our job easier
- While custom headers can be useful for a few reasons, the most common purpose is pretending to be a browser
    - This is typically done by setting the `'User-Agent'` field of the headers, although it depends on the site
    - Some are more rigorous than others and you need to pass a set of believable headers to get content back

Example-2: Sample Solution (Cont.)

```
import requests
from bs4 import BeautifulSoup
import pandas as pd


i=1
url=f'https://guba.eastmoney.com/list,zssh000001_{i}.html'


# Get the site HTML with header.

# The most common purpose to add headers is pretending to be a browser. Here are sample headers.
my_headers = {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "accept-encoding": "gzip, deflate, br",
    "accept-language": "en-US,en;q=0.8",
    "upgrade-insecure-requests": "1",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) \
        AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36",
}

response = requests.get(url, headers=my_headers)
response
```

```
<Response [200]>
```

Example-2: Sample Solution (Cont.)

```
# Parse HTML with BeautifulSoup.
bs = BeautifulSoup(response.content, 'html.parser')
divs = bs.find_all(name='tr', class_='listitem') # After inspecting the website, we find the the rule of the content

rows = []
for div in divs:
    # read
    read = div.find(name='div', class_='read').text
    # reply
    reply = div.find(name='div', class_='reply').string
    # title
    title = div.find(name='div', class_='title').string
    # author
    author = div.find(name='div', class_='author').string
    # update
    update = div.find(name='div', class_='update').string

    # Create a dictionary of this info
    post_dict = {'read': read, 'reply': reply, 'title': title, 'author': author, 'update':update}
    # Add it to our list of rows
    rows.append(post_dict)

# Now just transform our rows into a DataFrame
post_df = pd.DataFrame(rows)
post_df
```