Lecture S1. Web Page and Crawler

Instructor: Luping Yu

April 25, 2023

Web Page

Before we start writing code, we need to understand a little bit about the structure of a web page.

When we visit a web page, our web browser makes a request to a web server. This request is called a `GET` request, since we're getting files from the server. The server then sends back files that tell our browser how to render the page for us. These files will typically include:

- HTML — the main content of the page.
- CSS — used to add styling to make the page look nicer.
- JS — Javascript files add interactivity to web pages.

After our browser receives all the files, it **renders** the page and displays it to us.

There's a lot that happens behind the scenes to render a page nicely, but we don't need to worry about most of it when we're web scraping. When we perform web scraping, we’re interested in the main content of the web page, so we look primarily at the `HTML`.

What is HTML?

- HTML stands for Hyper Text Markup Language
- HTML is the standard **markup language** for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of **elements**
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

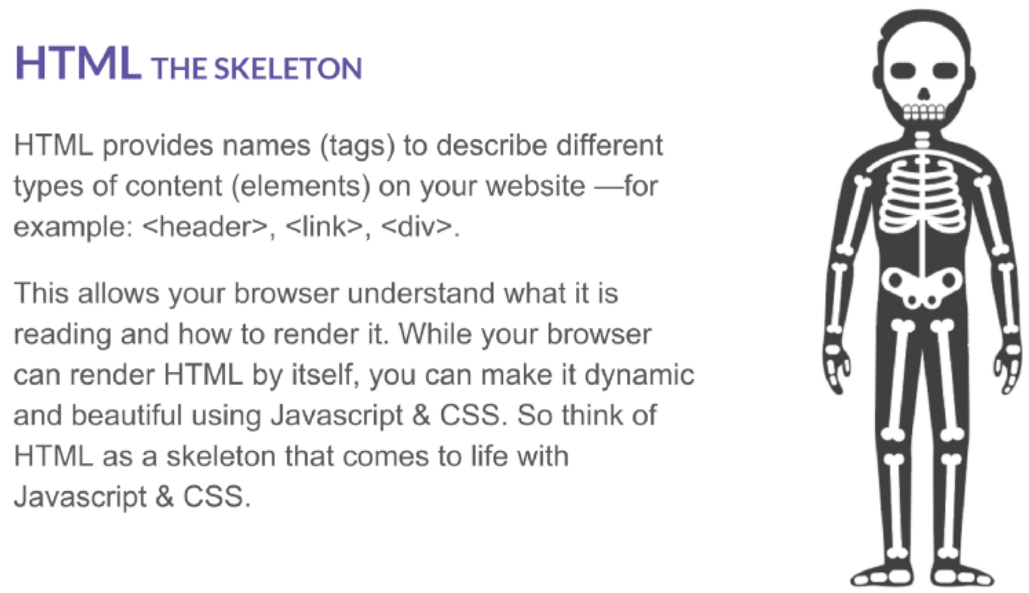
How to View HTML Source?

View HTML Source Code:

- Right-click in an HTML page and select "View Page Source" (in Chrome) or "View Source" (in Edge), or similar in other browsers. This will open a window containing the HTML source code of the page.

Chrome Developer Tools `F12` :

- More Tools ---> Developer Tools
- Chrome DevTools is a set of web developer tools built directly into the Google Chrome browser.



HTML THE SKELETON

HTML provides names (tags) to describe different types of content (elements) on your website —for example: <header>, <link>, <div>.

This allows your browser understand what it is reading and how to render it. While your browser can render HTML by itself, you can make it dynamic and beautiful using Javascript & CSS. So think of HTML as a skeleton that comes to life with Javascript & CSS.

A Simple HTML Document (a.k.a. "Page")

In [7]:

```
%%HTML

<!DOCTYPE html>
<html>

    <head>
        <title>My page title</title>
    </head>

    <body>
        <h1>Hello!</h1>
        <p>My first paragraph.</p>
    </body>

</html>
```

Hello!

My first paragraph.

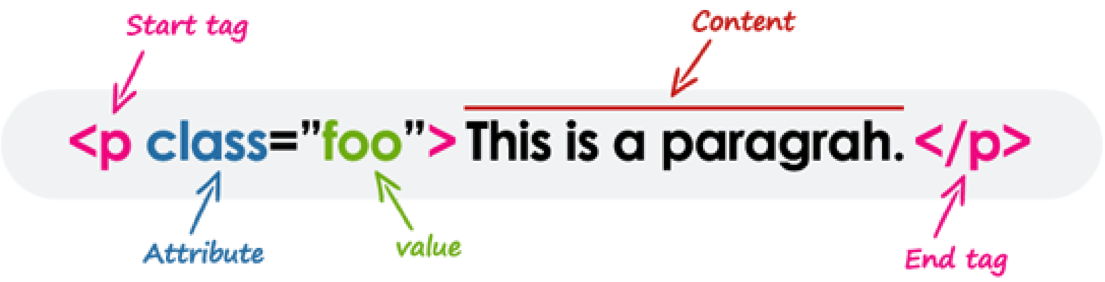
Example Explained

- The `<!DOCTYPE html>` declaration defines that this document is an HTML document

- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the HTML page
- The `<title>` element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The `<body>` element defines the document's body, and is a container for all the **visible** contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

HTML Element

An HTML `element` is defined by a **start tag**, some **content**, and an **end tag**:



The HTML **element** is everything from the start tag to the end tag:

In [9]:
```
%%HTML

<h1>My First Heading</h1>
<p>My first paragraph.</p>
```

My First Heading

My first paragraph.

---

1. Empty HTML Elements

- Empty elements (also called self-closing or void elements) are not container tags.
- A typical example of an empty element, is the `<br>` element, which represents a line break. Some other common empty elements are `<img>` , `<input>` , etc.

In [13]:
```
%%HTML

<p>This paragraph contains <br> a line break.</p>
<img src="https://www.xmu.edu.cn/images/logo2.png" alt="xmu">
<input type="text" name="username">
```

This paragraph contains
a line break.

2. Nesting HTML Elements

- Most HTML elements can contain any number of further elements, which are, in turn, made up of tags, attributes, and content or other elements.
- The following example shows some elements nested inside the `<p>` element.

In [14]:

```
%%HTML

<p>Here is some <b>bold</b> text.</p>
<p>Here is some <em>emphasized</em> text.</p>
<p>Here is some <mark>highlighted</mark> text.</p>
```

Here is some **bold** text.

Here is some *emphasized* text.

Here is some highlighted text.

3. HTML Links

HTML links are defined with the `<a>` tag:

- The link's destination is specified in the `href` attribute.
- `Attributes` are used to provide **additional information** about HTML elements.

In [15]:

```
%%HTML

<a href="https://sm.xmu.edu.cn/">This is a link</a>
```

This is a link

4. Writing Comments in HTML

- Comments are usually added with the purpose of making the source code easier to understand.
- You can also comment out part of your HTML code for debugging purpose.
- An HTML comment begins with `<!--, and ends with -->`, as shown in the example below:

In [16]:

```
%%HTML

<!-- This is an HTML comment -->
<!-- This is a multi-line HTML comment
     that spans across more than one line -->
<p>This is a normal piece of text.</p>
```

This is a normal piece of text.

---

5. HTML Elements Types

The basic elements of an HTML page are:

- A text header, denoted using the `<h1>, <h2>, <h3>, <h4>, <h5>, <h6>` tags.
- A paragraph, denoted using the `<p>` tag.
- A link, denoted using the `<a>` (anchor) tag.
- A list, denoted using the `<ul>` (unordered list), `<ol>` (ordered list) and `<li>` (list element) tags.
- An image, denoted using the `<img>` tag
- A divider, denoted using the `<div>` tag
- A text span, denoted using the `<span>` tag

Elements can be placed in two distinct groups: **block level** and **inline level** elements. The former make up the document's structure, while the latter dress up the contents of a block.

- A block element occupies 100% of the available width and it is rendered with a line break before and after. Whereas, an inline element will take up only as much space as it needs.
    - The most commonly used block-level elements are `<div>, <p>, <h1>` through `<h6>, <form>, <ol>, <ul>, <li>`, and so on. Whereas, the commonly used inline-level elements are `<img>, <a>, <span>, <strong>, <b>, <em>, <i>, <code>, <input>, <button>`, etc.
- The block-level elements should not be placed within inline-level elements. For example, the `<p>` element should not be placed inside the `<b>` element.

HTML Attributes

`Attributes` define **additional characteristics or properties** of the element such as width and height of an image.

- Attributes are always specified in the start tag (or opening tag) and usually consists of name/value pairs like `name="value"`.
    - Some attributes are required for certain elements. For instance, an `<img>` tag must contain a src and alt attributes.

Let's take a look at some examples of the attributes usages:

In [21]:
```
%%HTML

<img src="https://www.xmu.edu.cn/images/logo2.png" width="200" height="10
<a href="https://www.google.com/" title="Search Engine">Google</a>
<input type="text" value="Guido van Rossum">
```

Google Guido van Rossum

In the above example `src` inside the `<img>` tag is an attribute and image path provided is its value. Similarly `href` inside the `<a>` tag is an attribute and the link provided is its value, and so on.

There are some **general purpose** attributes, such as `id, title, class, style`, etc. that you can use on the majority of HTML elements.

1.The `id` Attribute

The `id` attribute is used to give a unique identifier to an element within a document. This makes it easier to select the element using CSS or JavaScript.

- The `id` of an element must be **unique** within a single document.
  - No two elements in the same document can be named with the same id.
  - Each element can have only one id.

In [22]:
```
%%HTML

<input type="text" id="firstName">
<div id="container">Some content</div>
<p id="infoText">This is a paragraph.</p>
```

Some content

This is a paragraph.

---

2. The `class` Attribute

Like `id` attribute, the `class` attribute is also used to identify elements. But unlike `id`, the class attribute **does not have to be unique** in the document.

- This means you can apply the same class to multiple elements in a document.
- Any style rules that are written to that class will be applied to all the elements having that class.

In [23]:

```
%%HTML

<input type="text" class="highlight">
<div class="box highlight">Some content</div>
<p class="highlight">This is a paragraph.</p>
```

Some content

This is a paragraph.

---

3. The `style` Attribute

HTML is quite limited when it comes to the presentation of a web page. It was originally designed as a simple way of presenting information.

CSS (Cascading Style Sheets) was introduced in December 1996 by the World Wide Web Consortium (W3C) to provide a better way to style HTML elements.

The `style` attribute allows you to specify CSS styling rules such as color, font, border, etc. directly within the element. Let's check out an example to see how it works:

In [24]:

```
%%HTML

<p style="color: blue;">This is a paragraph.</p>
<div style="border: 1px solid red;">Some content</div>
```

This is a paragraph.

Some content

---

4. HTML Attributes Types

Each element can also have attributes - each element has a different set of attributes relevant to the element.

There are a few global elements, the most common of them are:

- `id` - Denotes the unique ID of an element in a page. Used for locating elements by using links, JavaScript, and more.
- `class` - Denotes the CSS class of an element.
- `style` - Denotes the CSS styles to apply to an element.

Crawler

Some websites offer data sets that are downloadable in CSV format, or accessible via an **Application Programming Interface (API)**. But many websites with useful data don't offer these convenient options.

Web scraping is the process of **gathering information** from the Internet. Even copying and pasting the lyrics of your favorite song is a form of web scraping!

The words "web scraping" usually refer to a process that involves automation. Some websites don't like it when automatic scrapers gather their data, while others don't mind.

Common tools:

- `requests`
- `BeautifulSoup`
- `Selenium`
- `Pandas.read_html()`

How Does Web Scraping Work?

When we scrape the web, we write code that sends a request to the server that's hosting the page we specified. The server will return the source code — HTML, mostly — for the page (or pages) we requested.

So far, we're essentially doing the same thing a web browser does — sending a server request with a specific URL and asking the server to return the code for that page.

But unlike a web browser, our web scraping code won't interpret the page's source code and display the page visually. Instead, we'll write some custom code that filters through the page's source code looking for specific elements we've specified, and extracting whatever content we've instructed it to extract.

For example, if we wanted to get all of the data from inside a table that was displayed on a web page, our code would be written to go through these steps in sequence:

- Request the content (source code) of a specific URL from the server.
- Download the content that is returned.
- Identify the elements of the page that are part of the table we want.
- Extract and (if necessary) reformat those elements into a dataset we can analyze or use in whatever way we require.

Python and `BeautifulSoup` have built-in features designed to make this relatively straightforward.

The requests library

Now that we understand the structure of a web page, it's time to get into the fun part: scraping the content we want!

The first thing we'll need to do to scrape a web page is to download the page. We can download pages using the Python `requests` library.

The requests library will make a `GET` request to a web server, which will download the HTML contents of a given web page for us.

Let's try downloading a simple sample website:

In [25]:

```
import requests

page = requests.get("https://dataquestio.github.io/web-scraping-pages/sim

page
```

Out[25]:

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

A `status_code` of 200 means that the page downloaded successfully.

We won't fully dive into status codes here, but a status code starting with a 2 generally indicates success, and a code starting with a 4 or a 5 indicates an error.

We can print out the HTML content of the page using the content property:

In [26]:

```
page.content
```

Out[26]:

```
b'<!DOCTYPE html>\n<html>\n    <head>\n        <title>A simple example p
age</title>\n    </head>\n    <body>\n        <p>Here is some simple con
tent for this page.</p>\n    </body>\n</html>'
```

Parsing a page with BeautifulSoup

As you can see above, we now have downloaded an HTML document.

We can use the `BeautifulSoup` library to parse this document, and extract the text from the `p tag`. If we want to extract a single tag, we can use the `find_all` method, which will find all the instances of a tag on a page.

We first have to import the library, and create an instance of the `BeautifulSoup` class to parse our document:

In [27]:

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(page.content, 'html.parser')

soup
```

Out[27]:

```
<!DOCTYPE html>

<html>
<head>
<title>A simple example page</title>
</head>
<body>
<p>Here is some simple content for this page.</p>
</body>
</html>
```

In [28]:

```
soup.find_all('p')
```

Out[28]:

```
[<p>Here is some simple content for this page.</p>]
```

Note that `find_all` returns a list, so we'll have to loop through, or use list indexing, it to extract text:

In [29]:

```
soup.find_all('p')[0].get_text()
```

Out[29]:

```
'Here is some simple content for this page.'
```

Searching for tags by class and id

We introduced classes and ids earlier, but it probably wasn't clear why they were useful.

`Classes` and `ids` are used by CSS to determine which HTML elements to apply certain styles to. But when we're scraping, we can also use them to specify the elements we want to scrape.

To illustrate this principle, we'll work with another sample website:

In [31]:

```
page = requests.get("https://dataquestio.github.io/web-scraping-pages/ids
soup = BeautifulSoup(page.content, 'html.parser')
soup
```

Out[31]:

```
<html>
<head>
<title>A simple example page</title>
</head>
<body>
<div>
<p class="inner-text first-item" id="first">
                First paragraph.
            </p>
<p class="inner-text">
                Second paragraph.
            </p>
</div>
<p class="outer-text first-item" id="second">
<b>
                First outer paragraph.
            </b>
</p>
<p class="outer-text">
<b>
                Second outer paragraph.
            </b>
</p>
</body>
</html>
```

Now, we can use the `find_all` method to search for items by class or by id. Let's look for any tag that has the class `outer-text`:

In [32]:

```
soup.find_all(class_='outer-text')
```

Out[32]:

```
[<p class="outer-text first-item" id="second">
 <b>
                 First outer paragraph.
             </b>
 </p>,
 <p class="outer-text">
 <b>
                 Second outer paragraph.
             </b>
 </p>]
```

We can also search for elements by `id`:

In [33]:

```
soup.find_all(id="first")
```

Out[33]:

```
[<p class="inner-text first-item" id="first">
                 First paragraph.
             </p>]
```

Using CSS Selectors

We can also search for items using `CSS` selectors. These selectors are how the CSS language allows developers to specify HTML tags to style. Here are some examples:

- `p a` — finds all a tags inside of a p tag.

- `body p a` — finds all a tags inside of a p tag inside of a body tag.
- `html body` — finds all body tags inside of an html tag.
- `p.outer-text` — finds all p tags with a class of outer-text.
- `p#first` — finds all p tags with an id of first.
- `body p.outer-text` — finds any p tags with a class of outer-text inside of a body tag.

`BeautifulSoup` objects support searching a page via CSS selectors using the `select` method. We can use CSS selectors to find all the `p` tags in our page that are inside of a `div` like this:

In [34]:

```
soup.select("div p")
```

Out[34]:

```
[<p class="inner-text first-item" id="first">
                 First paragraph.
             </p>,
 <p class="inner-text">
                 Second paragraph.
             </p>]
```

Example 1. Downloading weather data

We now know enough to proceed with extracting information about the local weather from the National Weather Service website.

The first step is to find the page we want to scrape. We'll extract weather information about downtown San Francisco from this page. The page has information about the extended forecast for the next week, including time of day, temperature, and a brief description of the conditions.

Specifically, let's extract data about the extended forecast.

---

1. Exploring page structure with Chrome DevTools

The first thing we'll need to do is inspect the page using Chrome Devtools. If you’re using another browser, Firefox and Safari have equivalents.

You can start the developer tools in Chrome by clicking View -> Developer -> Developer Tools. Make sure the Elements panel is highlighted.

The elements panel will show you all the HTML tags on the page, and let you navigate through them. It's a really handy feature!

We can then scroll up in the elements panel to find the "outermost" element that contains all of the text that corresponds to the extended forecasts. In this case, it's a `div` tag with the id `seven-day-forecast`.

If we click around on the console, and explore the div, we'll discover that each forecast item (like "Tonight", "Thursday", and "Thursday Night") is contained in a `div` with the class `tombstone-container`.

2. Time to Start Scraping!

We now know enough to download the page and start parsing it. In the below code, we will:

- Download the web page containing the forecast.
- Create a BeautifulSoup class to parse the page.
- Find the div with id seven-day-forecast, and assign to seven_day
- Inside seven_day, find each individual forecast item.
- Extract and print the first forecast item.

In [35]:

```
page = requests.get("https://forecast.weather.gov/MapClick.php?lat=37.777
soup = BeautifulSoup(page.content, 'html.parser')

seven_day = soup.find(id="seven-day-forecast")

forecast_items = seven_day.find_all(class_="tombstone-container")

tonight = forecast_items[0]

tonight
```

Out[35]:

```
<div class="tombstone-container">
<p class="period-name">Tonight<br/><br/></p>
<p><img alt="Tonight: A 20 percent chance of rain.  Mostly cloudy, with
a low around 45. West wind 13 to 16 mph, with gusts as high as 20 mph. "
class="forecast-icon" src="newimages/medium/nra20.png" title="Tonight: A
20 percent chance of rain.  Mostly cloudy, with a low around 45. West wi
nd 13 to 16 mph, with gusts as high as 20 mph. "/></p><p class="short-de
sc">Slight Chance<br/>Rain</p><p class="temp temp-low">Low: 45 °F</p></d
iv>
```

3. Extracting information from the page

As we can see, inside the forecast item tonight is all the information we want. There are four pieces of information we can extract:

- The name of the forecast item — in this case, Tonight.
- The description of the conditions — this is stored in the title property of img.
- A short description of the conditions — in this case, Mostly Clear.
- The temperature low — in this case, 49 degrees.

We'll extract the name of the forecast item, the short description, and the temperature first, since they're all similar:

In [36]:

```
period = tonight.find(class_="period-name").get_text()
short_desc = tonight.find(class_="short-desc").get_text()
temp = tonight.find(class_="temp").get_text()
print(period)
print(short_desc)
print(temp)
```

```
Tonight
Slight ChanceRain
Low: 45 °F
```

Now, we can extract the `title` attribute from the `img` tag. To do this, we just treat the `BeautifulSoup` object like a dictionary, and pass in the attribute we want as a key:

In [37]:

```
img = tonight.find("img")
desc = img['title']
print(desc)
```

```
Tonight: A 20 percent chance of rain.  Mostly cloudy, with a low around
45. West wind 13 to 16 mph, with gusts as high as 20 mph.
```

---

4. Extracting all the information from the page

Now that we know how to extract each individual piece of information, we can combine our knowledge with CSS selectors and list comprehensions to **extract everything at once**.

In the below code, we will:

- Select all items with the class `period-name` inside an item with the class `tombstone-container` in `seven_day`.
- Use a list comprehension to call the `get_text` method on each `BeautifulSoup` object.

In [38]:

```
period_tags = seven_day.select(".tombstone-container .period-name")
periods = [pt.get_text() for pt in period_tags]
periods
```

Out[38]:

```
['Tonight',
 'Tuesday',
 'TuesdayNight',
 'Wednesday',
 'WednesdayNight',
 'Thursday',
 'ThursdayNight',
 'Friday',
 'FridayNight']
```

As we can see above, our technique gets us each of the period names, in order. We can apply the same technique to get the other three fields:

In [39]:

```
short_descs = [sd.get_text() for sd in seven_day.select(".tombstone-conta
temps = [t.get_text() for t in seven_day.select(".tombstone-container .te
descs = [d["title"] for d in seven_day.select(".tombstone-container img")
print(short_descs)
print(temps)
print(descs)
```

```
['Slight ChanceRain', 'Sunny', 'Mostly Clear', 'Sunny', 'Mostly Clearand
Breezythen MostlyClear', 'Sunny', 'Mostly Clear', 'Sunny', 'Partly Cloud
y']
['Low: 45 °F', 'High: 57 °F', 'Low: 46 °F', 'High: 59 °F', 'Low: 45 °F',
'High: 63 °F', 'Low: 49 °F', 'High: 68 °F', 'Low: 50 °F']
['Tonight: A 20 percent chance of rain.  Mostly cloudy, with a low aroun
d 45. West wind 13 to 16 mph, with gusts as high as 20 mph. ', 'Tuesday:
Sunny, with a high near 57. West wind 7 to 12 mph increasing to 13 to 18
mph in the afternoon. Winds could gust as high as 23 mph. ', 'Tuesday Ni
ght: Mostly clear, with a low around 46. West northwest wind 14 to 18 mp
h, with gusts as high as 23 mph. ', 'Wednesday: Sunny, with a high near
59. West wind 11 to 21 mph, with gusts as high as 26 mph. ', 'Wednesday
Night: Mostly clear, with a low around 45. Breezy, with a north northwes
t wind 17 to 22 mph decreasing to 9 to 14 mph after midnight. Winds coul
d gust as high as 28 mph. ', 'Thursday: Sunny, with a high near 63.', 'T
hursday Night: Mostly clear, with a low around 49.', 'Friday: Sunny, wit
h a high near 68.', 'Friday Night: Partly cloudy, with a low around 5
0.']
```

5. Combining our data into a Pandas Dataframe

We can now combine the data into a `Pandas DataFrame` and analyze it.

In [40]:

```python
import pandas as pd

weather = pd.DataFrame({"period": periods,
                        "short_desc": short_descs,
                        "temp": temps,
                        "desc":descs
})

weather
```

Out[40]:

	period	short_desc	temp	desc
0	Tonight	Slight ChanceRain	Low: 45 °F	Tonight: A 20 percent chance of rain. Mostly ...
1	Tuesday	Sunny	High: 57 °F	Tuesday: Sunny, with a high near 57. West wind...
2	TuesdayNight	Mostly Clear	Low: 46 °F	Tuesday Night: Mostly clear, with a low around...
3	Wednesday	Sunny	High: 59 °F	Wednesday: Sunny, with a high near 59. West wi...
4	WednesdayNight	Mostly Clearand Breezythen MostlyClear	Low: 45 °F	Wednesday Night: Mostly clear, with a low arou...
5	Thursday	Sunny	High: 63 °F	Thursday: Sunny, with a high near 63.
6	ThursdayNight	Mostly Clear	Low: 49 °F	Thursday Night: Mostly clear, with a low aroun...
7	Friday	Sunny	High: 68 °F	Friday: Sunny, with a high near 68.
8	FridayNight	Partly Cloudy	Low: 50 °F	Friday Night: Partly cloudy, with a low around...

Example 2. Pandas Web Scraping

Pandas makes it easy to scrape a `table` tag on a web page. You can use the function `read_html(url)` to get webpage contents.

It's only suitable for fetching `Table` type data, then let's see what kind of pages meet the conditions?

HTML Tables

HTML tables allow web developers to arrange data into rows and columns:

In [41]:

```
%%HTML

<table>
  <tr>
    <th>Company</th>
    <th>Contact</th>
    <th>Country</th>
  </tr>
  <tr>
    <td>Apple</td>
    <td>Tim Cook</td>
    <td>United States</td>
  </tr>
  <tr>
    <td>Tencent</td>
    <td>Pony Ma</td>
    <td>China</td>
  </tr>
</table>
```

Company	Contact	Country
Apple	Tim Cook	United States
Tencent	Pony Ma	China

Open the web version with a browser, F12 view the HTML structure of the web page, you will find that the structure of the eligible web page has a common feature.

If you find `Table` format, you can use `pd.read_html( )`

Sina finance - Institutional ownership

Take the aggregate shareholding data of Sina Financial institutions as an example:

In [1]:

```
url = 'https://vip.stock.finance.sina.com.cn/q/go.php/vComStockHold/kind/

df = pd.read_html(url)[0]

df
```

Out[1]:

	代码	简称	截至日期	家数	本期持股数(万股)	持股占已流通A股比例(%)	同上期增减(万股)	持股比例(%)	上期家数	明细
0	600012	皖通高速	2023-03-31	1	312.9698	0.19	-665.5320	0.59	2	+展开明细
1	600256	广汇能源	2023-03-31	3	26318.4082	4.01	7927.6651	2.80	2	+展开明细
2	600313	农发种业	2023-03-31	4	2431.9931	2.25	-402.8900	2.62	4	+展开明细
3	600318	新力金融	2023-03-31	1	731.8953	1.43	238.1553	0.96	1	+展开明细
4	600389	江山股份	2023-03-31	1	217.7107	0.71	-76.8536	0.96	1	+展开明细
5	600509	天富能源	2023-03-31	4	3852.4000	3.35	1133.7300	2.36	4	+展开明细
6	600557	康缘药业	2023-03-31	3	2669.4919	4.57	1171.4653	2.56	2	+展开明细
7	600566	济川药业	2023-03-31	1	632.4300	0.69	123.9789	0.55	1	+展开明细
8	600603	广汇物流	2023-03-31	2	2126.0165	1.69	-1031.9026	2.52	4	+展开明细
9	600861	北京城乡	2023-03-31	6	3385.5455	10.69	359.2874	9.55	5	+展开明细
10	561	烽火电子	2023-03-31	3	1029.9700	1.70	309.3000	1.19	2	+展开明细
11	610	西安旅游	2023-03-31	1	168.1100	0.71	30.0800	0.58	1	+展开明细
12	633	合金投资	2023-03-31	4	1263.6400	3.28	236.3800	2.67	3	+展开明细

	代码	简称	截至日期	家数	本期持股数(万股)	持股占已流通A股比例(%)	同上期增减(万股)	持股比例(%)	上期家数	明细
13	661	长春高新	2023-03-31	4	1592.3522	3.93	620.6031	2.40	2	+展开明细
14	735	罗牛山	2023-03-31	1	799.1143	0.69	30.3989	0.67	1	+展开明细
15	739	普洛药业	2023-03-31	4	8722.9498	7.40	-466.4628	7.80	4	+展开明细
16	915	华特达因	2023-03-31	2	820.0385	3.50	28.2400	3.38	2	+展开明细
17	929	兰州黄河	2023-03-31	2	252.4900	1.36	121.8000	0.70	1	+展开明细
18	969	安泰科技	2023-03-31	2	583.7817	0.57	221.7900	0.35	1	+展开明细
19	600436	片仔癀	2023-03-31	1	812.7227	1.35	-4.3900	1.35	1	+展开明细
20	600031	三一重工	2023-03-31	1	4571.0327	0.54	-223.2000	0.56	1	+展开明细
21	600976	健民集团	2023-03-31	6	1678.2520	10.94	233.6757	9.42	5	+展开明细
22	600918	中泰证券	2023-03-31	2	10496.6189	1.51	467.7975	1.44	2	+展开明细
23	300927	江天化学	2023-03-31	1	100.0992	0.69	100.0992	0.00	0	+展开明细
24	688722	同益中	2023-03-31	1	220.6135	0.98	5.0881	0.96	1	+展开明细
25	2176	江特电机	2023-03-31	5	9423.9136	5.52	9423.9136	0.00	0	+展开明细
26	2191	劲嘉	2023-03-31	1	647.5759	0.44	-2659.7400	2.25	3	+展开明细

	代码	简称	截至日期	家数	本期持股数(万股)	持股占已流通A股比例(%)	同上期增减(万股)	持股比例(%)	上期家数	明细
		股份								
27	688276	百克生物	2023-03-31	1	207.1771	0.50	-698.0485	2.19	3	+展开明细
28	2262	恩华药业	2023-03-31	1	1506.0000	1.49	398.9600	1.10	1	+展开明细
29	300680	隆盛科技	2023-03-31	5	1373.0820	5.94	-635.2159	8.69	6	+展开明细
30	2334	英威腾	2023-03-31	4	5602.6844	7.16	4877.3445	0.93	1	+展开明细
31	300624	万兴科技	2023-03-31	2	269.3060	2.07	177.8146	0.70	1	+展开明细
32	601718	际华集团	2023-03-31	1	1567.7885	0.36	-612.9900	0.50	1	+展开明细
33	2709	天赐材料	2023-03-31	4	7140.7478	3.71	2051.9341	2.64	3	+展开明细
34	688676	金盘科技	2023-03-31	3	918.6769	2.15	93.9094	1.94	3	+展开明细
35	2697	红旗连锁	2023-03-31	2	2202.6652	1.62	1467.0420	0.54	1	+展开明细
36	2788	鹭燕医药	2023-03-31	1	130.5320	0.34	130.5320	0.00	0	+展开明细
37	688004	博汇科技	2023-03-31	2	63.3445	1.12	63.3445	0.00	0	+展开明细
38	2546	新联电子	2023-03-31	1	360.4400	0.43	10.4400	0.42	1	+展开明细
39	300263	隆华	2023-03-31	1	441.1100	0.49	-393.8200	0.92	1	+展开明细

	代码	简称	截至日期	家数	本期持股数(万股)	持股占已流通A股比例(%)	同上期增减(万股)	持股比例(%)	上期家数	明细
		科技								

In [2]:

```
df = pd.DataFrame()

for i in range(1, 10):
    url = 'https://vip.stock.finance.sina.com.cn/q/go.php/vComStockHold/k
    df = pd.concat([df, pd.read_html(url)[0]])
    print("Page %s completed!" % i)

df
```

```
Page 1 completed!
Page 2 completed!
Page 3 completed!
Page 4 completed!
Page 5 completed!
Page 6 completed!
Page 7 completed!
Page 8 completed!
Page 9 completed!
```

Out[2]:

	代码	简称	截至日期	家数	本期持股数(万股)	持股占已流通A股比例(%)	同上期增减(万股)	持股比例(%)	上期家数	明细
0	600012	皖通高速	2023-03-31	1	312.9698	0.19	-665.5320	0.59	2	+展开明细
1	600256	广汇能源	2023-03-31	3	26318.4082	4.01	7927.6651	2.80	2	+展开明细
2	600313	农发种业	2023-03-31	4	2431.9931	2.25	-402.8900	2.62	4	+展开明细
3	600318	新力金融	2023-03-31	1	731.8953	1.43	238.1553	0.96	1	+展开明细
4	600389	江山股份	2023-03-31	1	217.7107	0.71	-76.8536	0.96	1	+展开明细
...	...	...	...	...	...	...	...	...	...	...
30	301363	美好医疗	2023-03-31	7	823.6502	2.03	-20.5426	2.08	8	+展开明细
31	301049	超越科技	2023-03-31	1	11.6000	0.12	11.6000	0.00	0	+展开明细
32	688700	东威科技	2023-03-31	1	184.6832	1.25	184.6832	0.00	0	+展开明细
33	688522	纳睿雷达	2023-03-31	9	651.1404	4.21	651.1404	0.00	0	+展开明细
34	688475	萤石网络	2023-03-31	5	1155.4342	2.05	1155.4342	0.00	0	+展开明细

75 rows × 10 columns