In [1]:
```
from dsc80_utils import *
```

Lecture 10 – Web Scraping

DSC 80, Fall 2025

Announcements 📣

- Midterm Exam *next* Thursday!
- Exam review session **this** Thursday!
- See Campuswire for details on both.

Agenda 📆

- Review: Accessing HTML.
- HTML basics.
- Parsing HTML using BeautifulSoup.
- Example: Scraping quotes.
- Example: Scraping the HDSI faculty page.

Review: Accessing HTML

Making requests

Goal: Access information about HDSI faculty members from the HDSI Faculty page.

Let's start by making a GET request to the HDSI Faculty page and see what the resulting HTML looks like.

In [2]:
```
import requests
```

In [3]:
```
fac_response = requests.get('https://datascience.ucsd.edu/faculty/', verify=
fac_response
```

```
/Users/eldridge/workbench/dsc80/.venv/lib/python3.12/site-packages/urllib3/c
onnectionpool.py:1099: InsecureRequestWarning: Unverified HTTPS request is b
eing made to host 'datascience.ucsd.edu'. Adding certificate verification is
strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usa
ge.html#tls-warnings
  warnings.warn(
```

Out[3]:
```
<Response [200]>
```

In [4]:
```python
fac_text = fac_response.text
len(fac_text)
```

Out[4]: 258316

In [5]:
```python
print(fac_text[:1000])
```


```
<!DOCTYPE html><html lang="en-US"><head><meta charset="UTF-8"><meta name="vi
ewport" content="width=device-width, initial-scale=1.0" /><meta http-equiv
="X-UA-Compatible" content="IE=edge"><link rel="profile" href="https://gmpg.
org/xfn/11" /><title>Faculty &#8211; Halıcıoğlu Data Science Institute &#821
1; UC San Diego</title> <script defer src="data:text/javascript;base64,CiAgI
CAgICAgICAgICAgICAgICAgICAgICAgICAvKiBZb3UgY2FuIGFkZCBtb3JlIGNvbmZpZ3VyYXRpb
24gb3B0aW9ucyB0byB3ZWJmb250bG9hZGVyIGJ5IHByZXZpb3VzbHkgZGVmaW5pbmcgdGhlIFdlY
kZvbnRDb25maWcgd2l0aCB5b3VyIG9wdGlvbnMgKi8KICAgICAgICAgICAgICAgICAgICAgICAgI
CAgIGlmICggdHlwZW9mIFdlYkZvbnRDb25maWcgPT09ICJ1bmRlZmluZWQiICkgewogICAgICAgI
CAgICAgICAgICAgICAgICAgICAgIFdlYkZvbnRDb25maWcgPSBuZXcgT2JqZWN0KCk7CiAgI
CAgICAgICAgICAgICAgICAgICAgICB9CiAgICAgICAgICAgICAgICAgICAgICAgICAgICBXZ
WJGb250Q29uZmlnWydnb29nbGUnXSA9IHtmYW1pbGllczogWydSb2JvdG86NDAwLDUwMCcsICdkb
3N0OjQwMCw3MDAnLCAnVGVrbzo3MDAnXX07CgogICAgICAgICAgICAgICAgICAgICAgICAgICAgK
GZ1bmN0aW9uKCk
```


Wow, that is gross looking! 😰

- It is **raw** HTML, which web browsers use to display websites.
- The information we are looking for – faculty information – is in there somewhere, but we have to search for it and extract it, which we wouldn't have to do if we had an API.
- We'll now look at how HTML documents are structured and how to extract information from them.

Best practices for scraping

1. **Send requests slowly** and be upfront about what you are doing!
2. Respect the policy published in the page's `robots.txt` file.
   - Many sites have a `robots.txt` file in their root directory, which contains a policy that allows or disallows automatic access to their site.
   - If there isn't one, like in Project 3, use a 0.5 second delay between requests.
3. Don't spoof your User-agent (i.e. don't try to trick the server into thinking you are a person).
4. Read the Terms of Service for the site and follow it.

Consequences of irresponsible scraping

If you make too many requests:

- The server may block your IP Address.
- You may take down the website.

Loading [MathJax]/extensions/Safe.js

- A journalist scraped and accidentally took down the Cook County Inmate Locater.
- As a result, inmate's families weren't able to contact them while the site was down.

Summary: APIs vs. scraping

- **APIs** are made by organizations that host data.
  - For example, X (formally known as Twitter) has an API.
  - APIs provide a code-friendly way to access data.
  - Usually, APIs give us back data as JSON objects.

- **Scraping** is the act of emulating a web browser to access its source code.
  - As you'll see in Lab 5, it's not technically supported by most organizations.
  - When scraping, you get back data as HTML and have to parse that HTML to extract the information you want.

The anatomy of HTML documents

What is HTML?

- HTML (HyperText Markup Language) is **the** basic building block of the internet.
- It defines the content and layout of a webpage, and as such, it is what you get back when you scrape a webpage.
- See this tutorial for more details.

For instance, here's the content of a very basic webpage.

In [6]:
```
!cat data/lec10_ex1.html
```

```
<html>
  <head>
    <title>Page title</title>
  </head>

  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is <b>another</b> paragraph.</p>
  </body>
</html>
```

Using `IPython.display.HTML`, we can render it directly in our notebook.

In [7]:
```
from IPython.display import HTML
HTML(filename=Path('data') / 'lec10_ex1.html')
```

Out[7]:

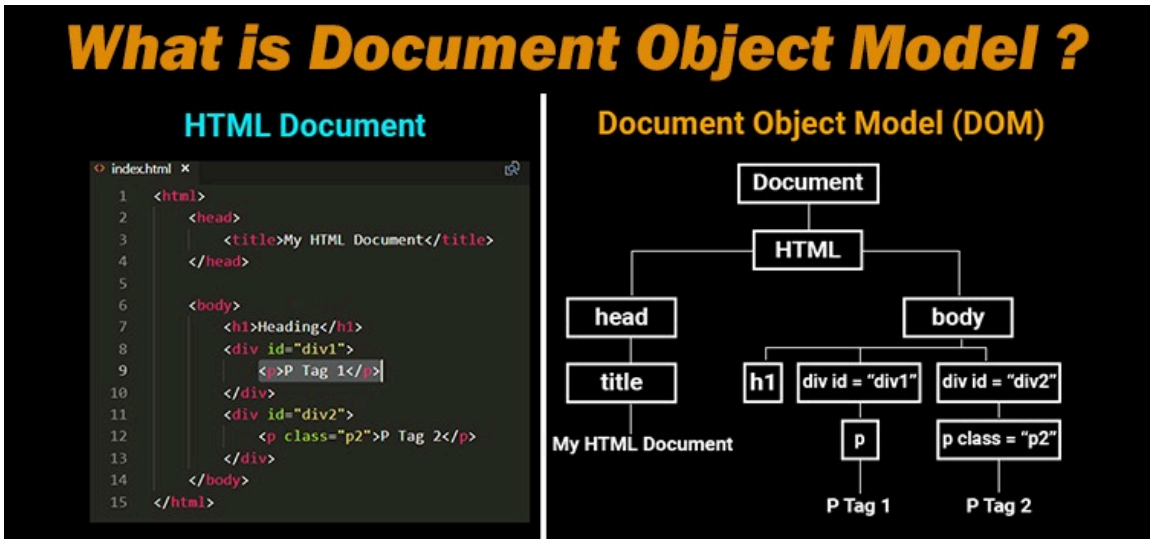
This is a heading

This is a paragraph.

This is **another** paragraph.

The anatomy of HTML documents

- **HTML document**: The totality of markup that makes up a webpage.
- **Document Object Model (DOM)**: The internal representation of an HTML document as a hierarchical **tree** structure.
- **HTML element**: An object in the DOM, such as a paragraph, header, or title.
- **HTML tags**: Markers that denote the **start** and **end** of an element, such as `<p>` and `</p>`.



(source)

Useful tags to know

Element	Description
`<html>`	the document
`<head>`	the header
`<body>`	the body

Element	Description
`<div>`	a logical division of the document
`<span>`	an *inline* logical division
`<p>`	a paragraph
`<a>`	an anchor (hyperlink)
`<h1>, <h2>, ...`	header(s)
`<img>`	an image

There are many, many more, but these are by far the most common. See this article for examples.

Example: Images and hyperlinks

Tags can have **attributes**, which further specify how to display information on a webpage.

For instance, `<img>` tags have `src` and `alt` attributes (among others):

```
<img src="king-selfie.png" alt="A photograph of King Triton."
width=500>
```

Hyperlinks have `href` attributes:

```
Click <a href="https://practice.dsc80.com">this link</a> to access
past exams.
```

What do you think this webpage looks like?

In [8]:
```
!cat data/lec10_ex2.html
```

```
<html>
  <head>
    <title>Project 4A and 4B – DSC 80, Spring 2024</title>
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/boo
tstrap.min.css"
      rel="stylesheet"
    />
  </head>

  <body>
    <h1>Project Overview</h1>
    <img src="../imgs/platter.png" width="200" alt="My dinner last night." /
>
    <p>
      When the project is released, you can start it by
      <a href="https://github.com/dsc-courses/dsc80-2024-wi/"
        >public GitHub repo</a
      >.
    </p>
    <center>
      <h3>
        Note that you'll have to submit your notebook as a PDF and a link to
        your website.
      </h3>
    </center>
  </body>
</html>
```

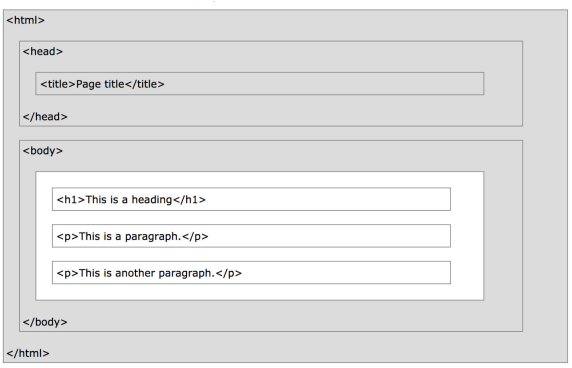
The `<div>` tag

```
<div style="background-color:lightblue">
  <h3>This is a heading</h3>
  <p>This is a paragraph.</p>
</div>
```

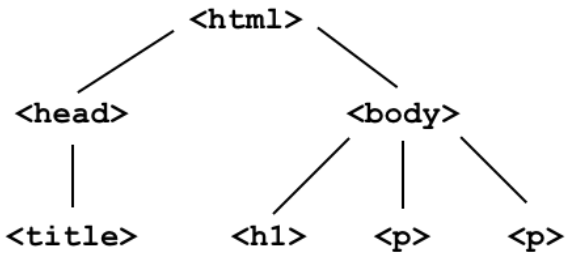
- The `<div>` tag defines a division or a "section" of an HTML document.
  - Think of a `<div>` as a "cell" in a Jupyter Notebook.
- The `<div>` element is often used as a container for other HTML elements to style them with CSS or to perform operations involving them using JavaScript.
- `<div>` elements often have attributes, **which are important when scraping**!

Document trees

Under the document object model (DOM), HTML documents are trees. In DOM trees, child nodes are **ordered**.



What does the DOM tree look like for this document?



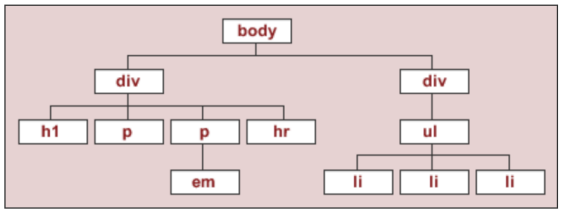
Parsing HTML using Beautiful Soup

Beautiful Soup 🍜

- Beautiful Soup 4 is a Python HTML parser.
  - To "parse" means to "extract meaning from a sequence of symbols".
- **Warning**: Beautiful Soup 4 and Beautiful Soup 3 work differently, so make sure you are using and looking at documentation for Beautiful Soup 4.

Example HTML document

To start, we'll work with the source code for an HTML page with the DOM tree shown below:



The string `html_string` contains an HTML "document".

In [9]:
```
html_string = '''
<html>
    <body>
        <h1 id="content">
            <h1>Heading here</h1>
```

```
        <p>My First paragraph</p>
        <p>My <em>second</em> paragraph</p>
        <hr>
      </div>
      <div id="nav">
        <ul>
          <li>item 1</li>
          <li>item 2</li>
          <li>item 3</li>
        </ul>
      </div>
    </body>
</html>
'''.strip()
```

In [10]:
```
HTML(html_string)
```

Out[10]:

Heading here

My First paragraph

My *second* paragraph

- item 1
- item 2
- item 3

`BeautifulSoup` objects

`bs4.BeautifulSoup` takes in a string or file-like object representing HTML (`markup`) and returns a **parsed** document.

In [11]:
```
import bs4
```

In [12]:
```
bs4.BeautifulSoup?
```

Loading [MathJax]/extensions/Safe.js

```
Init signature:
bs4.BeautifulSoup(
    markup='',
    features=None,
    builder=None,
    parse_only=None,
    from_encoding=None,
    exclude_encodings=None,
    element_classes=None,
    **kwargs,
)
Docstring:
A data structure representing a parsed HTML or XML document.

Most of the methods you'll call on a BeautifulSoup object are inherited from
PageElement or Tag.

Internally, this class defines the basic interface called by the
tree builders when converting an HTML/XML document into a data
structure. The interface abstracts away the differences between
parsers. To write a new tree builder, you'll need to understand
these methods as a whole.

These methods will be called by the BeautifulSoup constructor:
  * reset()
  * feed(markup)

The tree builder may call these methods from its feed() implementation:
  * handle_starttag(name, attrs) # See note about return value
  * handle_endtag(name)
  * handle_data(data) # Appends to the current data node
  * endData(containerClass) # Ends the current data node

No matter how complicated the underlying parser is, you should be
able to build a tree using 'start tag' events, 'end tag' events,
'data' events, and "done with data" events.

If you encounter an empty-element tag (aka a self-closing tag,
like HTML's <br> tag), call handle_starttag and then
handle_endtag.
Init docstring:
Constructor.

:param markup: A string or a file-like object representing
 markup to be parsed.

:param features: Desirable features of the parser to be
 used. This may be the name of a specific parser ("lxml",
 "lxml-xml", "html.parser", or "html5lib") or it may be the
 type of markup to be used ("html", "html5", "xml"). It's
 recommended that you name a specific parser, so that
 Beautiful Soup gives you the same results across platforms
 and virtual environments.

:param builder: A TreeBuilder subclass to instantiate (or
 instance to use) instead of looking one up based on
```

```
 `features`. You only need to use this if you've implemented a
 custom TreeBuilder.

:param parse_only: A SoupStrainer. Only parts of the document
 matching the SoupStrainer will be considered. This is useful
 when parsing part of a document that would otherwise be too
 large to fit into memory.

:param from_encoding: A string indicating the encoding of the
 document to be parsed. Pass this in if Beautiful Soup is
 guessing wrongly about the document's encoding.

:param exclude_encodings: A list of strings indicating
 encodings known to be wrong. Pass this in if you don't know
 the document's encoding but you know Beautiful Soup's guess is
 wrong.

:param element_classes: A dictionary mapping BeautifulSoup
 classes like Tag and NavigableString, to other classes you'd
 like to be instantiated instead as the parse tree is
 built. This is useful for subclassing Tag or NavigableString
 to modify default behavior.

:param kwargs: For backwards compatibility purposes, the
 constructor accepts certain keyword arguments used in
 Beautiful Soup 3. None of these arguments do anything in
 Beautiful Soup 4; they will result in a warning and then be
 ignored.

 Apart from this, any keyword arguments passed into the
 BeautifulSoup constructor are propagated to the TreeBuilder
 constructor. This makes it possible to configure a
 TreeBuilder by passing in arguments, not just by saying which
 one to use.
File:           ~/workbench/dsc80/.venv/lib/python3.12/site-packages/bs4/__i
nit__.py
Type:           type
Subclasses:     BeautifulStoneSoup
```

Normally, we pass the result of a GET request to `bs4.BeautifulSoup`, but here we will pass our hand-crafted `html_string`.

In [13]:
```
soup = bs4.BeautifulSoup(html_string)
soup
```

Out[13]:
```
<html>
<body>
<div id="content">
<h1>Heading here</h1>
<p>My First paragraph</p>
<p>My <em>second</em> paragraph</p>
<hr/>
</div>
<div id="nav">
<ul>
<li>item 1</li>
<li>item 2</li>
<li>item 3</li>
</ul>
</div>
</body>
</html>
```

In [14]:
```
type(soup)
```

Out[14]:
```
bs4.BeautifulSoup
```

`BeautifulSoup` objects have several useful attributes, e.g. `text`:

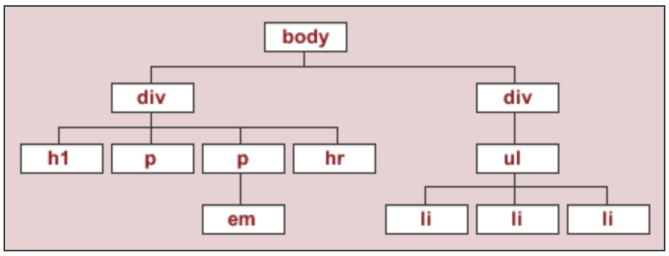
In [15]:
```
print(soup.text)
```

```
Heading here
My First paragraph
My second paragraph



item 1
item 2
item 3
```

Traversing through `descendants`

The `descendants` attribute traverses a `BeautifulSoup` tree using **depth-first traversal**.

Why depth-first? Elements closer to one another on a page are more likely to be related than elements further away.



In [16]:
```
soup.descendants
```

Out[16]:
```
<generator object Tag.descendants at 0x116180790>
```

In [17]:
```
for child in soup.descendants:
#     print(child) # What would happen if we ran this instead?
    if isinstance(child, str):
        continue
    print(child.name)
```

```
html
body
div
h1
p
p
em
hr
div
ul
li
li
li
```

Finding elements in a tree

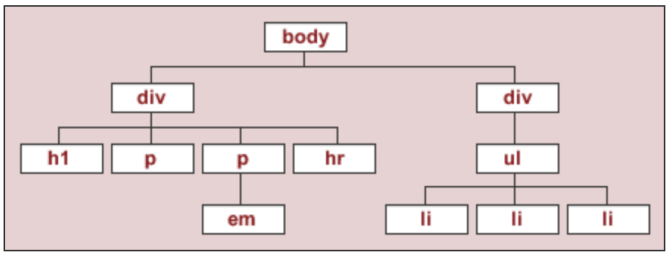
Practically speaking, you will not use the `descendants` attribute (or the related `children` attribute) directly very often. Instead, you will use the following methods:

- `soup.find(tag)`, which finds the **first** instance of a tag (the first one on the page, i.e. the first one that DFS sees).
  - More general: `soup.find(name=None, attrs={}, recursive=True, text=None, **kwargs)`.
- `soup.find_all(tag)` will find **all** instances of a tag.

`find` finds tags!

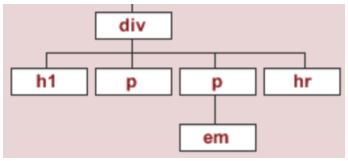
Using `find`

Let's try and extract the first `<div>` subtree.



In [18]:
```
soup.find('div')
```

Out[18]:
```
<div id="content">
<h1>Heading here</h1>
<p>My First paragraph</p>
<p>My <em>second</em> paragraph</p>
<hr/>
</div>
```



Let's try and find the `<div>` element that has an `id` attribute equal to `'nav'`.

In [19]:
```
soup.find('div', attrs={'id': 'nav'})
```

Out[19]:
```
<div id="nav">
<ul>
<li>item 1</li>
<li>item 2</li>
<li>item 3</li>
</ul>
</div>
```

`find` will return the first occurrence of a tag, regardless of its depth in the tree.

In [20]:
```
# The ul child is not at the top of the tree, but we can still find it.
soup.find('ul')
```

Out[20]:
```
<ul>
<li>item 1</li>
<li>item 2</li>
<li>item 3</li>
</ul>
```

Using `find_all`

`find_all` returns a list of all matches.

In [21]:
```
soup.find_all('div')
```

Out[21]:
```
[<div id="content">
 <h1>Heading here</h1>
 <p>My First paragraph</p>
 <p>My <em>second</em> paragraph</p>
 <hr/>
 </div>,
 <div id="nav">
 <ul>
 <li>item 1</li>
 <li>item 2</li>
 <li>item 3</li>
 </ul>
 </div>]
```

In [22]:
```
soup.find_all('li')
```

Out[22]:
```
[<li>item 1</li>, <li>item 2</li>, <li>item 3</li>]
```

In [23]:
```
[x.text for x in soup.find_all('li')]
```

Out[23]:
```
['item 1', 'item 2', 'item 3']
```

Node attributes

- The `text` attribute of a tag element gets the text between the opening and closing tags.
- The `attrs` attribute of a tag element lists all of its attributes.
- The `get` method of a tag element **gets the value of an attribute**.

In [24]:
```
soup.find('p')
```

Out[24]:
```
<p>My First paragraph</p>
```

In [25]:
```
soup.find('p').text
```

Out[25]:
```
'My First paragraph'
```

In [26]:
```
soup.find('div')
```

Out[26]:
```
<div id="content">
<h1>Heading here</h1>
<p>My First paragraph</p>
<p>My <em>second</em> paragraph</p>
<hr/>
</div>
```

In [27]:
```
soup.find('div').text
```

Out[27]:
```
'\nHeading here\nMy First paragraph\nMy second paragraph\n\n'
```

In [28]:
```
soup.find('div').attrs
```

Out[28]:
```
{'id': 'content'}
```

In [29]:
```
soup.find('div').get('id')
```

Out[29]:
```
'content'
```

The `get` method must be called directly on the node that contains the attribute you're looking for.

In [30]:
```
soup
```

Out[30]:
```
<html>
<body>
<div id="content">
<h1>Heading here</h1>
<p>My First paragraph</p>
<p>My <em>second</em> paragraph</p>
<hr/>
</div>
<div id="nav">
<ul>
<li>item 1</li>
<li>item 2</li>
<li>item 3</li>
</ul>
</div>
</body>
</html>
```

In [31]:
```
# While there are multiple 'id' attributes, none of them are in the <html> t
soup.get('id')
```

In [32]:
```
soup.find('div').get('id')
```

Out[32]:
```
'content'
```

Question 🤔

Consider the following HTML document, which represents a webpage containing the top few songs with the most streams on Spotify today in Canada.

```
<head>
    <title>3*Canada-2022-06-04</title>
</head>
<body>
    <h1>Spotify Top 3 - Canada</h1>
    <table>
        <tr class='heading'>
            <th>Rank</th>
            <th>Artist(s)</th>
            <th>Song</th>
```

```
            </tr>
            <tr class=1>
                <td>1</td>
                <td>Harry Styles</td>
                <td>As It Was</td>
            </tr>
            <tr class=2>
                <td>2</td>
                <td>Jack Harlow</td>
                <td>First Class</td>
            </tr>
            <tr class=3>
                <td>3</td>
                <td>Kendrick Lamar</td>
                <td>N95</td>
            </tr>
        </table>
</body>
```

Part 1: How many leaf nodes are there in the DOM tree of the previous document — that is, how many nodes have no children?

Part 2: What does the following line of code evaluate to?

```
len(soup.find_all("td"))
```

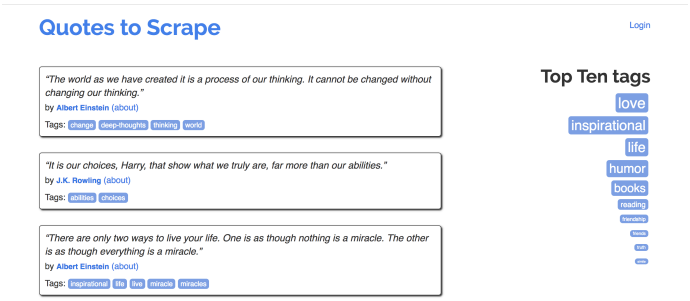
Part 3: What does the following line of code evaluate to?

```
soup.find("tr").get("class")
```

Example: Scraping quotes

Example: Scraping quotes

Consider quotes.toscrape.com.



Goal: Extract quotes (and relevant metadata) into a DataFrame.

Specifically, let's try to make a DataFrame that looks like the one below:

	quote	author	author_url	
0	“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”	Albert Einstein	https://quotes.toscrape.com/author/Albert-Einstein	change,deep-thoughts,thinking,world
1	“It is our choices, Harry, that show what we truly are, far more than our abilities.”	J.K. Rowling	https://quotes.toscrape.com/author/J-K-Rowling	abilities,choices
2	“There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.”	Albert Einstein	https://quotes.toscrape.com/author/Albert-Einstein	inspirational,life,live,mirac

Question 🤔

Ask an LLM to write code to scrape the first ten pages of quotes from https://quotes.toscrape.com/ into a DataFrame called `quotes_llm`. The first three rows of `quotes_llm` should have the three quotes above. The last row of `quotes_llm` should contain a quote from George R.R. Martin.

After having an LLM write code, paste it below and see if it works. If it doesn't work, try to adjust your prompt until it does. Once you have something that works, submit your final prompt and generated code to http://dsc80.com/q.

In [ ]:

The plan

Eventually, we will create a single function – `make_quote_df` – which takes in an integer `n` and returns a **DataFrame** with the quotes on the **first `n` pages** of quotes.toscrape.com.

To do this, we will define several helper functions:

- `download_page(i)`, which downloads a **single page** (page `i`) and returns a `BeautifulSoup` object of the response.
- `process_quote(div)`, which takes in a `<div>` tree corresponding to a **single quote** and returns a dictionary containing all of the relevant information for that quote.
- `process_page(divs)`, which takes in a list of `<div>` trees corresponding to a **single page** and returns a DataFrame containing all of the relevant information for all quotes on that page.

Key principle: some of our helper functions will make **requests**, and others will **parse**, but none will do both!

- Easier to debug and catch errors.
- Avoids unnecessary requests.

Downloading a single page

In [33]:
```
def download_page(i):
    url = f'https://quotes.toscrape.com/page/{i}'
    request = requests.get(url)
    return bs4.BeautifulSoup(request.text)
```

In `make_quote_df`, we will call `download_page` repeatedly – once for `i=1`, once for `i=2`, ..., `i=n`. For now, we will work with just page 1 (chosen arbitrarily).

In [34]:
```
soup = download_page(1)
```

Parsing a single page

Let's look at the page's source code (right click the page and click "Inspect" in Chrome) to find where the quotes in the page are located.

In [35]:
```
divs = soup.find_all('div', class_='quote')
# Shortcut for the following, just for when the attribute key is class:
```

```
# divs = soup.find_all('div', attrs={'class': 'quote'})
```

In [36]:
```
divs[0]
```

Out[36]:
```
<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
<span class="text" itemprop="text">“The world as we have created it is a pr
ocess of our thinking. It cannot be changed without changing our thinkin
g.”</span>
<span>by <small class="author" itemprop="author">Albert Einstein</small>
<a href="/author/Albert-Einstein">(about)</a>
</span>
<div class="tags">
            Tags:
            <meta class="keywords" content="change,deep-thoughts,thinking,w
orld" itemprop="keywords"/>
<a class="tag" href="/tag/change/page/1/">change</a>
<a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
<a class="tag" href="/tag/thinking/page/1/">thinking</a>
<a class="tag" href="/tag/world/page/1/">world</a>
</div>
</div>
```

From this `<div>`, we can extract the quote, author name, author's URL, and tags.

In [37]:
```
divs[0]
```

Out[37]:
```
<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
<span class="text" itemprop="text">“The world as we have created it is a pr
ocess of our thinking. It cannot be changed without changing our thinkin
g.”</span>
<span>by <small class="author" itemprop="author">Albert Einstein</small>
<a href="/author/Albert-Einstein">(about)</a>
</span>
<div class="tags">
            Tags:
            <meta class="keywords" content="change,deep-thoughts,thinking,w
orld" itemprop="keywords"/>
<a class="tag" href="/tag/change/page/1/">change</a>
<a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
<a class="tag" href="/tag/thinking/page/1/">thinking</a>
<a class="tag" href="/tag/world/page/1/">world</a>
</div>
</div>
```

In [38]:
```
# The quote.
divs[0].find('span', class_='text').text
```

Out[38]:
```
'“The world as we have created it is a process of our thinking. It cannot b
e changed without changing our thinking.”'
```

In [39]:
```
# The author.
divs[0].find('small', class_='author').text
```

Out[39]:
```
'Albert Einstein'
```

In [40]:
```
# The URL for the author.
divs[0].find('a').get('href')
```

Out[40]:
```
'/author/Albert-Einstein'
```

In [41]:
```
# The quote's tags.
divs[0].find('meta', class_='keywords').get('content')
```

Out[41]:
```
'change,deep-thoughts,thinking,world'
```

Let's implement our next function, `process_quote`, which takes in a `<div>` corresponding to a single quote and returns a dictionary containing the quote's information.

Why use a dictionary? Passing `pd.DataFrame` a list of dictionaries is an easy way to create a DataFrame.

In [42]:
```
def process_quote(div):
    quote = div.find('span', class_='text').text
    author = div.find('small', class_='author').text
    author_url = 'https://quotes.toscrape.com' + div.find('a').get('href')
    tags = div.find('meta', class_='keywords').get('content')

    return {'quote': quote, 'author': author, 'author_url': author_url, 'tag
```

In [43]:
```
process_quote(divs[-1])
```

Out[43]:
```
{'quote': '“A day without sunshine is like, you know, night.”',
 'author': 'Steve Martin',
 'author_url': 'https://quotes.toscrape.com/author/Steve-Martin',
 'tags': 'humor,obvious,simile'}
```

Our last helper function will take in a **list** of `<div>`s, call `process_quote` on each `<div>` in the list, and return a **DataFrame**.

In [44]:
```
def process_page(divs):
    return pd.DataFrame([process_quote(div) for div in divs])
```

In [45]:
```
process_page(divs)
```

Out[45]:

	quote	author	author_url	
0	“The world as we have created it is a process ...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	change,deep-thought
1	“It is our choices, Harry, that show what we t...	J.K. Rowling	https://quotes.toscrape.com/author/J-K-Rowling	
2	“There are only two ways to live your life. On...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	inspirational,life,live,
...	...	...	...	
7	“I have not failed. I've just found 10,000 way...	Thomas A. Edison	https://quotes.toscrape.com/author/Thomas-A-Ed...	edison,failure,inspiratio
8	“A woman is like a tea bag; you never know how...	Eleanor Roosevelt	https://quotes.toscrape.com/author/Eleanor-Roo...	misattributed-e
9	“A day without sunshine is like, you know, nig...	Steve Martin	https://quotes.toscrape.com/author/Steve-Martin	humo

10 rows × 4 columns

Putting it all together

In [46]:
```
def make_quote_df(n):
    '''Returns a DataFrame containing the quotes on the first n pages of htt
    dfs = []
    for i in range(1, n+1):
        # Download page n and create a BeautifulSoup object.
        soup = download_page(i)

        # Create DataFrame using the information in that page.
        divs = soup.find_all('div', class_='quote')
        df = process_page(divs)

        # Append DataFrame to dfs.
        dfs.append(df)

    # Stitch all DataFrames together.
    return pd.concat(dfs).reset_index(drop=True)
```

In [47]:
```
quotes = make_quote_df(3)
quotes.head()
```

Out[47]:

	quote	author	author_url	
0	“The world as we have created it is a process ...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	thoughts
1	“It is our choices, Harry, that show what we t...	J.K. Rowling	https://quotes.toscrape.com/author/J-K-Rowling	a
2	“There are only two ways to live your life. On...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	inspirational,life,live,m
3	“The person, be it gentleman or lady, who has ...	Jane Austen	https://quotes.toscrape.com/author/Jane-Austen	aliteracy,books
4	“Imperfection is beauty, madness is genius and...	Marilyn Monroe	https://quotes.toscrape.com/author/Marilyn-Monroe	be-yourse

In [48]:
```
quotes[quotes['author'] == 'Albert Einstein']
```

Out[48]:

	quote	author	author_url	
0	“The world as we have created it is a process ...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	c thoughts,t
2	“There are only two ways to live your life. On...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	inspirational,life,live,mira
5	“Try not to become a man of success. Rather be...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	adulthood,s
12	“If you can't explain it to a six year old, yo...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	simplicit
26	“If you want your children to be intelligent, ...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	childr
28	“Logic will get you from A to Z; imagination w...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	

The elements in the `'tags'` column are all strings, but they look like lists. This is not ideal, as we will see shortly.

Example: Scraping the HDSI faculty page

Example: Scraping the HDSI faculty page

Let's try and extract a list of HDSI Faculty from datascience.ucsd.edu/faculty.

- As usual, we start by opening the page, right clicking somewhere on the page, and clicking "Inspect" in Chrome.
- As we can see, the HTML is much more complicated – this is usually the case for websites in the wild.

In [49]:
```
fac_response = requests.get('https://datascience.ucsd.edu/faculty/', verify=
fac_response
```

```
/Users/eldridge/workbench/dsc80/.venv/lib/python3.12/site-packages/urllib3/c
onnectionpool.py:1099: InsecureRequestWarning: Unverified HTTPS request is b
eing made to host 'datascience.ucsd.edu'. Adding certificate verification is
strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usa
ge.html#tls-warnings
  warnings.warn(
```

Out[49]:
```
<Response [200]>
```

In [50]:
```
soup = bs4.BeautifulSoup(fac_response.text)
```

It's not easy identifying which `<div>`s we want. The Inspect tool makes this easier, but it's good to verify that `find_all` is finding the right number of elements.

In [51]:
```
divs = soup.find_all(
    class_='vc_grid-item',
)
```

In [52]:
```
len(divs)
```

Out[52]:
```
65
```

Within here, we need to extract each faculty member's name. It seems like names are stored as text within the `<h4>` tag.

In [53]:
```
divs[0]
```

Out[53]:
```
<div class="vc_grid-item vc_clearfix col_1-5 vc_grid-item-zone-c-bottom vc_
visible-item vc_grid-term-faculty-fellows vc_grid-term-council vc_grid-term
-faculty"> <a class="anchor-link" id="ilkay-altintas" name="ilkay-altinta
s"></a><div class="vc_grid-item-mini vc_clearfix"><div class="vc_gitem-anim
ated-block"><div class="vc_gitem-zone vc_gitem-zone-a vc-gitem-zone-height-
mode-auto vc-gitem-zone-height-mode-auto-1-1" style="background-image: url
(https://datascience.ucsd.edu/wp-content/uploads/2022/10/ilkayaltintas_head
shot.jpg) !important;"><a class="vc_gitem-link vc-zone-link" href="https://
datascience.ucsd.edu/people/ilkay-altintas/"></a><img alt="Ilkay Altintas"
class="vc_gitem-zone-img" decoding="async" src="https://datascience.ucsd.ed
u/wp-content/uploads/2022/10/ilkayaltintas_headshot.jpg"/><div class="vc_gi
tem-zone-mini"></div></div></div><div class="vc_gitem-zone vc_gitem-zone-
c"><div class="vc_gitem-zone-mini"><div class="vc_gitem_row vc_row vc_gitem
-row-position-top"><div class="vc_col-sm-12 vc_gitem-col vc_gitem-col-align
-"><div class="vc_custom_heading vc_gitem-post-data vc_gitem-post-data-sour
ce-post_title"><h4 style="text-align: left"><a href="https://datascience.uc
sd.edu/people/ilkay-altintas/">Ilkay Altintas</a></h4></div><div class="vc_
gitem-align-left fields"><div class="field pendari_people_title">SDSC Chief
Data Science Officer &amp;amp; HDSI Founding Faculty Fellow</div></div><div cla
ss="excerpt"></div><div class="terms"> Faculty Fellows Council Faculty</div
></div></div></div></div></div></div>
```


In [54]:
```
divs[0].find('h4').text
```

Out[54]:
```
'Ilkay Altintas'
```

We can also extract job titles:

In [55]:
```
divs[0].find(class_='field').text
```

Out[55]:
```
'SDSC Chief Data Science Officer & HDSI Founding Faculty Fellow'
```

Let's create a DataFrame consisting of names and job titles for each faculty member.

In [56]:
```
names = [div.find('h4').text for div in divs]
names[:10]
```

Out[56]:
```
['Ilkay Altintas',
 'Tiffany Amariuta',
 'Mikio Aoi',
 'Ery Arias-Castro',
 'Vineet Bafna',
 'Mikhail Belkin',
 'Umesh Bellur',
 'Peter Chi',
 'Henrik Christensen',
 'Alex Cloninger']
```

In [57]:
```
titles = [div.find(class_='field').text for div in divs]
titles[:10]
```

Out[57]:
```
['SDSC Chief Data Science Officer & HDSI Founding Faculty Fellow',
 'Assistant Professor',
 'Assistant Professor',
 'Professor',
 'Professor',
 'Professor',
 'Visiting Professor, UCSD HDSI',
 'Associate Teaching Professor',
 'Distinguished Scientist, Professor',
 'Professor']
```

In [58]:
```
faculty = pd.DataFrame({
    'name': names,
    'title': titles,
})
faculty.head()
```

Out[58]:

	name	title
0	Ilkay Altintas	SDSC Chief Data Science Officer & HDSI Foundin...
1	Tiffany Amariuta	Assistant Professor
2	Mikio Aoi	Assistant Professor
3	Ery Arias-Castro	Professor
4	Vineet Bafna	Professor

Now we have a DataFrame!

In [59]:
```
faculty[faculty['title'].str.contains('Teaching') | faculty['title'].str.cor
```

Out[59]:

	name	title
7	Peter Chi	Associate Teaching Professor
14	Justin Eldridge	Associate Teaching Professor
15	Shannon Ellis	Associate Teaching Professor
...	...	...
39	Giorgio Quer	Lecturer
47	Jack Silberman	Lecturer
51	Janine Tiefenbruck	Lecturer

9 rows × 2 columns

What if we want to get faculty members' pictures?

In [60]:
```
from IPython.display import Image, display

def show_picture(name):
    idx = faculty[faculty['name'].str.lower().str.contains(name.lower())].ir
    display(Image(url=divs[idx].find('img')['src'], width=200, height=200))

show_picture('marina')
```

Question 🤔

Consider the following HTML document, which represents a webpage containing the top few songs with the most streams on Spotify today in Canada.

```
<head>
    <title>3*Canada-2022-06-04</title>
</head>
```

```
<body>
    <h1>Spotify Top 3 - Canada</h1>
    <table>
        <tr class='heading'>
            <th>Rank</th>
            <th>Artist(s)</th>
            <th>Song</th>
        </tr>
        <tr class=1>
            <td>1</td>
            <td>Harry Styles</td>
            <td>As It Was</td>
        </tr>
        <tr class=2>
            <td>2</td>
            <td>Jack Harlow</td>
            <td>First Class</td>
        </tr>
        <tr class=3>
            <td>3</td>
            <td>Kendrick Lamar</td>
            <td>N95</td>
        </tr>
    </table>
</body>
```

Web data in practice

The spread of true and false news online by Vosoughi et al. compared how true and false news spreads via Twitter:

> There is worldwide concern over false news and the possibility that it can influence political, economic, and social well-being. To understand how false news spreads, Vosoughi et al. used a data set of rumor cascades on Twitter from 2006 to 2017. About 126,000 rumors were spread by ~3 million people. False news reached more people than the truth; the top 1% of false news cascades diffused to between 1000 and 100,000 people, whereas the truth rarely diffused to more than 1000 people. Falsehood also diffused faster than the truth. The degree of novelty and the emotional reactions of recipients may be responsible for the differences observed.

To conduct this study, the authors used the Twitter API for accessing tweets and web-scraped fact-checking websites to verify whether news was false or not.

Summary, next time

- Beautiful Soup is an HTML parser that allows us to (somewhat) easily extract information from HTML documents.
  - `soup.find` and `soup.find_all` are the functions you will use most often.
- When writing scraping code:
  - Use "inspect element" to identify the names of tags and attributes that are relevant to the information you want to extract.
  - Separate your logic for making requests and for parsing.

Next time

Regular expressions!

In [ ]: