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

# For the JSON evaluation example.
def err():
    raise ValueError('i just deleted all your files lol 😂')
```

Lecture 9 – HTTP

DSC 80, Fall 2024

Agenda 📆

- Probabilistic Imputation
- Introduction to HTTP.
- Making HTTP requests.
- Data formats.
- APIs and web scraping.
- Midterm review.

Recall: Imputation with single values

- Imputing missing data in a column with the mean of the column:
  - faithfully reproduces the mean of the observed dataset,
  - reduces the variance, and
  - biases relationships between the column and other columns if the data are not MCAR.
- The same is true with other statistics (e.g. median and mode).

In [2]:
```
from lec08_utils import make_mcar, make_mar_on_cat, multiple_kdes

heights_path = Path('../lec08/data') / 'midparent.csv'
heights = pd.read_csv(heights_path).rename(columns={'childHeight': 'child'})[[
heights.head()

np.random.seed(42) # So that we get the same results each time (for lecture).
heights_mcar = make_mcar(heights, 'child', pct=0.5)
heights_mar = make_mar_on_cat(heights, 'child', 'gender', pct=0.5)
```

In [3]:
```
heights_mar
```

Loading [MathJax]/extensions/Safe.js

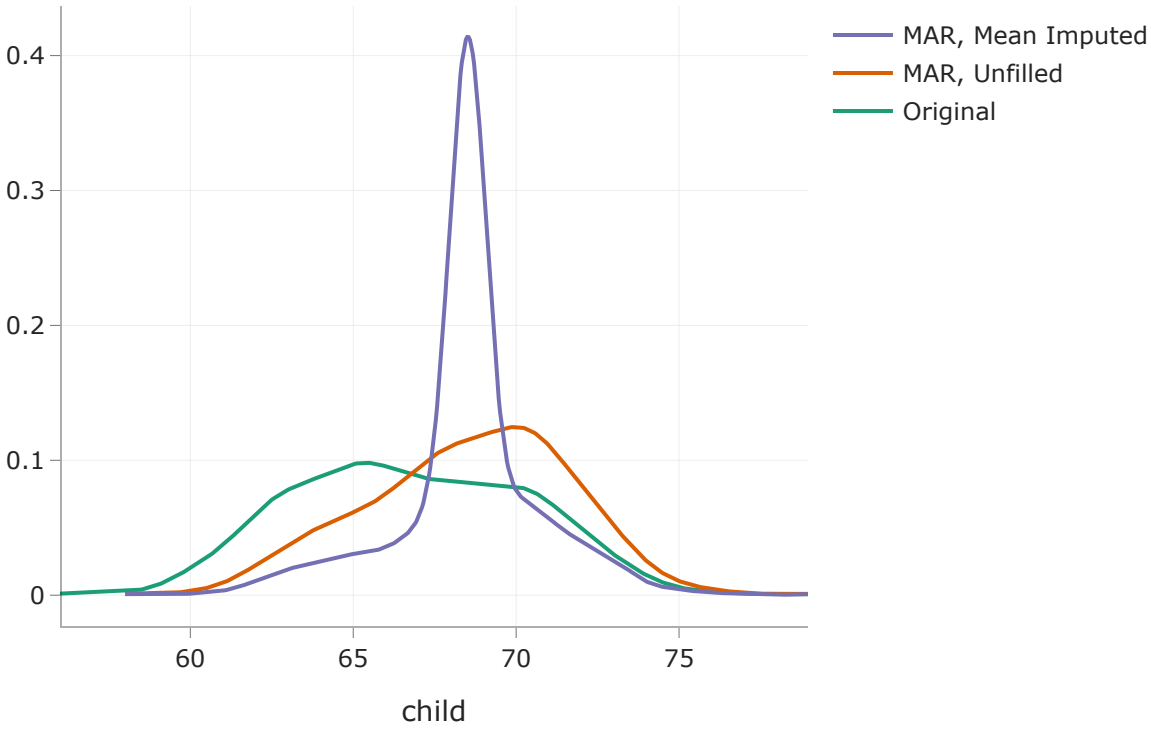
Out[3]:

	father	mother	gender	child
0	78.5	67.0	male	73.2
1	78.5	67.0	female	69.2
2	78.5	67.0	female	NaN
...	...	...	...	...
931	62.0	66.0	female	NaN
932	62.5	63.0	male	NaN
933	62.5	63.0	female	NaN

934 rows × 4 columns

In [4]:
```
heights_mar_filled = heights_mar.copy()
heights_mar_filled['child'] = heights_mar_filled['child'].fillna(heights_mar['
```

In [5]:
```
df_map = {'Original': heights, 'MAR, Unfilled': heights_mar, "MAR, Mean Impute
multiple_kdes(df_map)
```



Probabilistic imputation

Imputing missing values using distributions

Loading [MathJax]/extensions/Safe.js

- So far, each missing value in a column has been filled in with a constant value.
  - This creates "spikes" in the imputed distributions.

- **Idea**: We can **probabilistically** impute missing data from a distribution.
  - We can fill in missing data by drawing from the distribution of the **non-missing** data.
  - There are 5 missing values? Pick 5 values from the data that aren't missing.
  - How? Using `np.random.choice` or `.sample`.

- If the data are MCAR, then sample from the entire column of present values. If the data are MAR on some categorical column, then sample from the present values separately for each category.

Example: Probabilistic imputation in the MAR `heights` dataset

Let's use `transform` to call `prob_impute` separately on each `'gender'`.

In [6]:
```
def prob_impute(s):
    s = s.copy()

    # Step 1: Find the number of missing child heights for that gender.
    num_null = s.isna().sum()

    # Step 2: Sample num_null observed child heights for that gender.
    fill_values = np.random.choice(s.dropna(), num_null)

    # Step 3: Fill in missing values and return ser.
    s[s.isna()] = fill_values
    return s
```

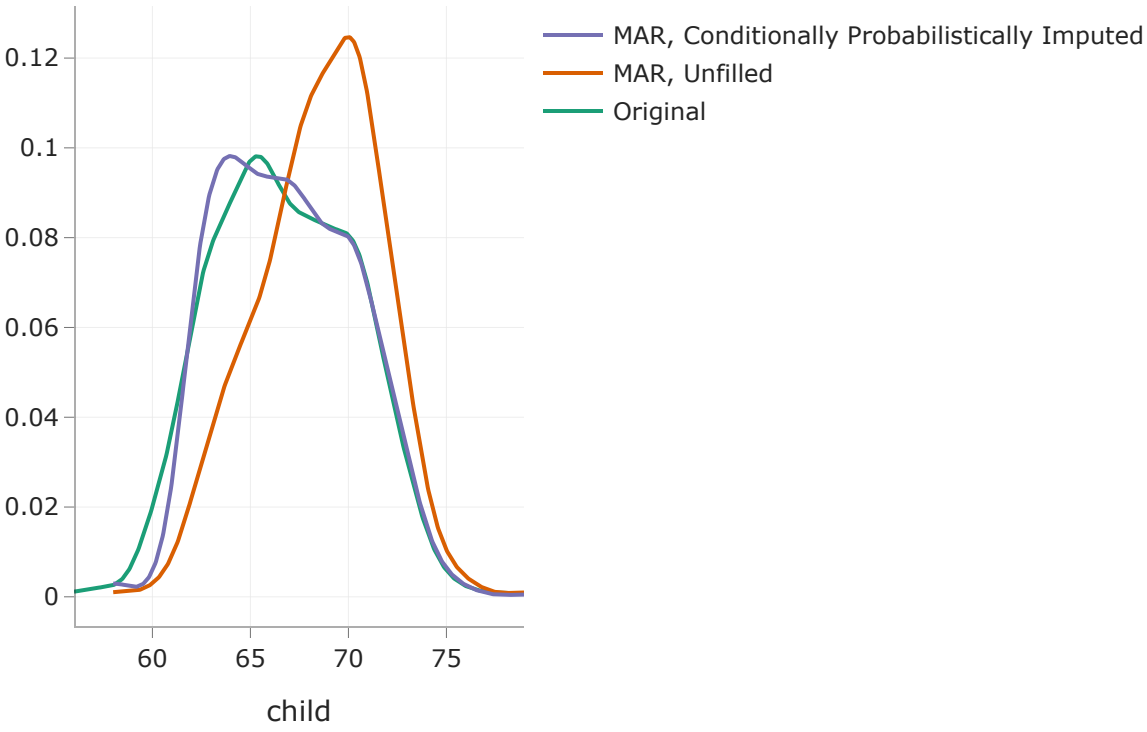
In [7]:
```
heights_mar_pfilled = heights_mar.copy()
heights_mar_pfilled['child'] = (
    heights_mar
    .groupby('gender')
    ['child']
    .transform(prob_impute)
)
heights_mar_pfilled['child'].head()
```

Out[7]:
```
0    73.2
1    69.2
2    62.0
3    62.5
4    73.5
Name: child, dtype: float64
```

In [8]:
```
df_map = {'Original': heights, 'MAR, Unfilled': heights_mar}
df_map['MAR, Conditionally Probabilistically Imputed'] = heights_mar_pfilled
```

Loading [MathJax]/extensions/Safe.js

```
multiple_kdes(df_map)
```



The **green distribution (conditional probabilistic imputation)** does the best job of approximating the **turqoise distribution (the full dataset with no missing values)**!

Remember that the graph above is interactive – you can hide/show lines by clicking them in the legend.

Observations

- With this technique, the missing values were filled in with observed values in the dataset.
- If a value was never observed in the dataset, it will never be used to fill in a missing value.
    - For instance, if the observed heights were 68, 69, and 69.5 inches, we will never fill a missing value with 68.5 inches even though it's a perfectly reasonable height.
- Solution? Create a histogram (with `np.histogram`) to bin the data, then sample from the histogram.
    - See Lab 5, Question 6.

Randomness

- Unlike mean imputation, probabilistic imputation is **random** – each time you run the cell in which imputation is performed, the results could be different.

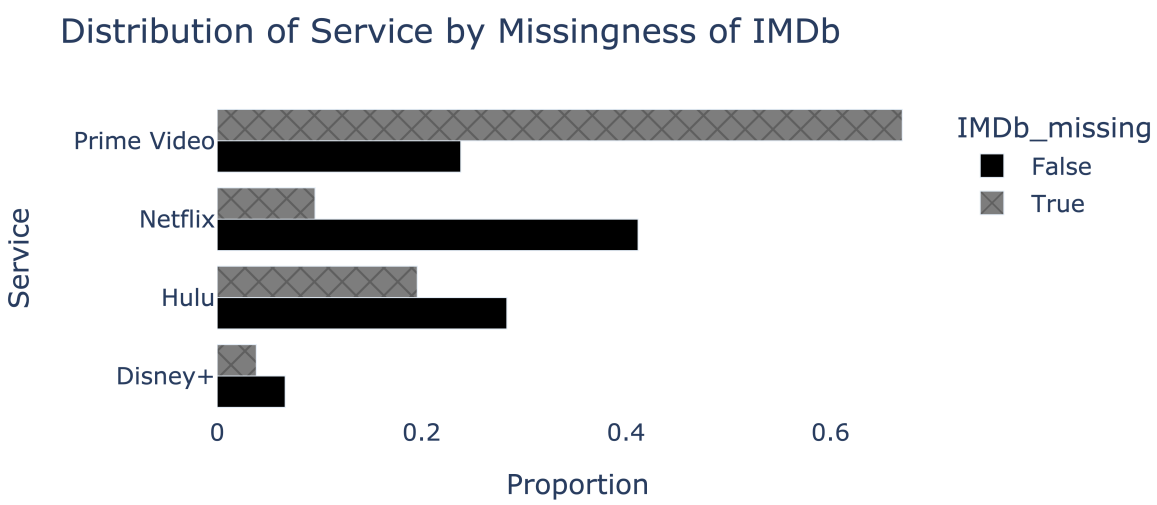
Loading [MathJax]/extensions/Safe.js

- If we're interested in estimating some population **parameter** given our (incomplete) sample, it's best not to rely on just a single random imputation.
- **Multiple imputation**: Generate multiple imputed datasets and aggregate the results!
  - Similar to bootstrapping.

Question 🤔

Taken from the Winter 2023 DSC 80 Midterm Exam.

To determine whether the missingness of `"IMDb"` depends on `"Service"`, we produce the following plot.



We'd like to fill in missing "IMDb" values in the fastest, most efficient way possible, such that the mean of the imputed "IMDb" column is as close to the true mean of the "IMDb" column in nature as possible. Which imputation technique should we use?

A. Unconditional mean imputation

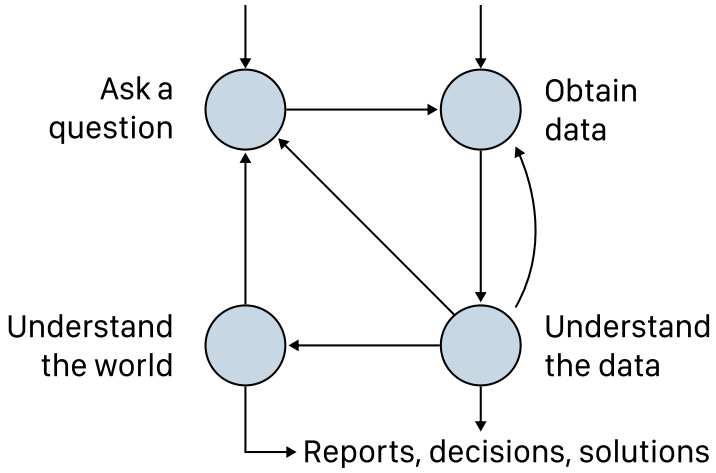
B. Mean imputation, conditional on "Service"

C. Unconditional probabilistic imputation

D. Probabilistic imputation, conditional on "Service"

Introduction to HTTP

Loading [MathJax]/extensions/Safe.js



Data sources

- Often, the data you need doesn't exist in "clean" `.csv` files.
- **Solution**: Collect your own data!
  - Design and administer your own survey or run an experiment.
  - Find related data on the internet.
- The internet contains **massive** amounts of historical record; for most questions you can think of, the answer exists somewhere on the internet.

Collecting data from the internet

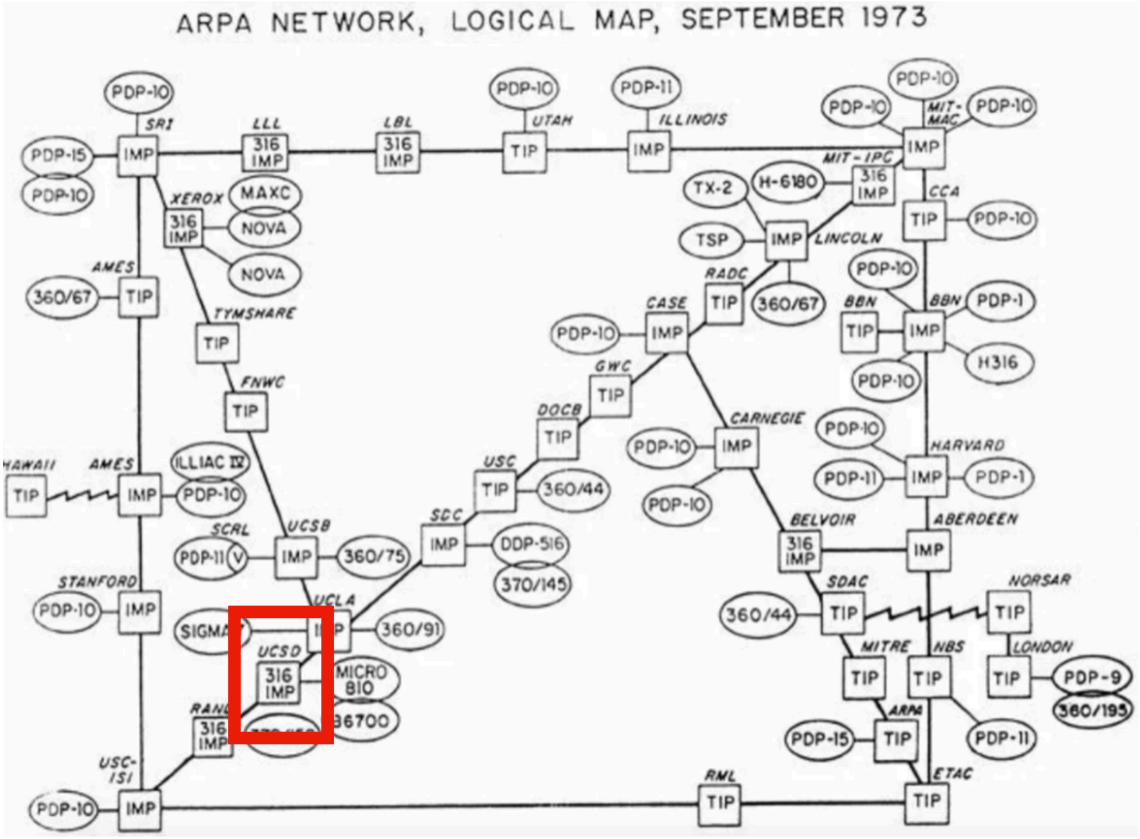
- There are two ways to programmatically access data on the internet:
  - through an API.
  - by scraping.
- We will discuss the differences between both approaches, but for now, the important part is that they **both use HTTP**.

HTTP

- HTTP stands for **Hypertext Transfer Protocol**.
  - It was developed in 1989 by Tim Berners-Lee (and friends).
- It is a **request-response** protocol.
  - Protocol = set of rules.
- HTTP allows...
  - computers to talk to each other over a network.

Loading [MathJax]/extensions/Safe.js to fetch data from "web servers."

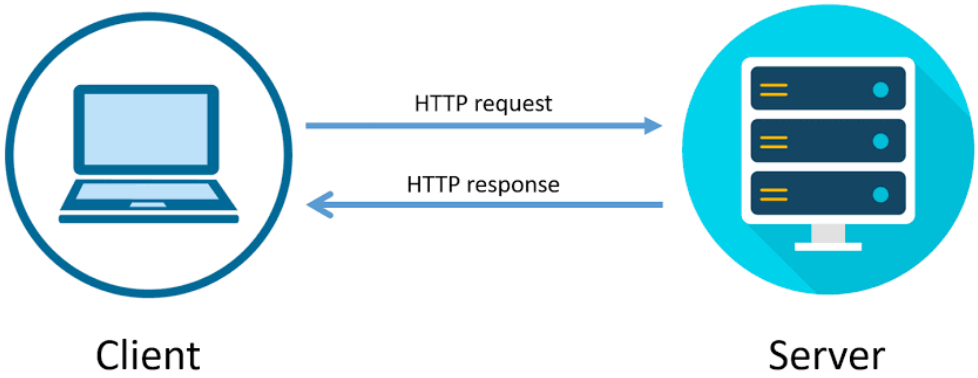
- The "S" in HTTPS stands for "secure".



UCSD was a node in ARPANET, the predecessor to the modern internet (source).

The request-response model

HTTP follows the **request-response** model.



- A **request** is made by the **client**.
- A **response** is returned by the **server**.
- **Example**: YouTube search 📽️.

Loading [MathJax]/extensions/Safe.js

- Consider the following URL: https://www.youtube.com/results?search_query=apple+vision+pro.
- Your web browser, a **client**, makes an HTTP **request** with a search query.
- The **server**, YouTube, is a computer that is sitting somewhere else.
- The server returns a **response** that contains the search results.
- Note: ?search_query=apple+vision+pro is called a "query string."

Request methods

The request methods you will use most often are `GET` and `POST`; see Mozilla's web docs for a detailed list of request methods.

- `GET` is used to request data **from** a specified resource.
- `POST` is used to **send** data to the server.
  - For example, uploading a photo to Instagram or entering credit card information on Amazon.

Example `GET` request

Below is an example `GET` HTTP request made by a browser when accessing datascience.ucsd.edu.

```
GET / HTTP/1.1
Connection: keep-alive
Host: datascience.ucsd.edu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36
sec-ch-ua: "Chromium";v="121", "Not A(Brand";v="99"
sec-ch-ua-platform: "macOS"
```

- The first line (`GET / HTTP/1.1`) is called the "request line", and the lines afterwards are called "header fields". Header fields contain metadata.
- We *could* also provide a "body" after the header fields.
- To see HTTP requests in Google Chrome, follow these steps.

Example `GET` response

The response below was generated by executing the request on the previous slide.

```
HTTP/1.1 200 OK
Date: Sun, 04 Feb 2024 17:35:01 GMT
Server: Apache
X-Powered-By: PHP/7.4.33
Link: <https://datascience.ucsd.edu/wp-json/>;
```

Loading [MathJax]/extensions/Safe.js

```
rel="https://api.w.org/"
Link: <https://datascience.ucsd.edu/wp-json/wp/v2/pages/113>;
rel="alternate"; type="application/json"
...

<html lang="en-US">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" 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>Halıcıoğlu Data Science Institute &#8211;UC San
Diego</title>
        <script>
...
```

Consequences of the request-response model

- When a request is sent to view content on a webpage, the server must:
  - process your request (i.e. prepare data for the response).
  - send content back to the client in its response.
- Remember, servers are computers.
  - Someone has to pay to keep these computers running.
  - **This means that every time you access a website, someone has to pay.**

Making HTTP requests

Making HTTP requests

There are (at least) two ways to make HTTP requests outside of a browser:

- From the command line, with `curl`.
- **From Python, with the `requests` package.**

Making HTTP requests using `requests`

- `requests` is a Python module that allows you to use Python to interact with the internet!
- There are other packages that work similarly (e.g. `urllib`), but `requests` is arguably the easiest to use.

```
ts
```

Loading [MathJax]/extensions/Safe.js

Example: `GET` requests via `requests`

For instance, let's access the source code of the UCSD homepage, https://ucsd.edu.

In [10]:
```
res = requests.get('https://ucsd.edu')
```

`res` is now a `Response` object.

In [11]:
```
res
```

Out[11]:
```
<Response [200]>
```

The `text` attribute of `res` is a string that containing the entire response.

In [12]:
```
type(res.text)
```

Out[12]:
```
str
```

In [13]:
```
len(res.text)
```

Out[13]:
```
63534
```

In [14]:
```
print(res.text[:1000])
```

Loading [MathJax]/extensions/Safe.js

```
<!DOCTYPE html>
<html lang="en">
  <head>
```

```
    <meta charset="utf-8"/>
    <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
    <meta content="width=device-width, initial-scale=1" name="viewport"/>
    <title>University of California San Diego</title>
    <meta content="University of California, San Diego" name="ORGANIZATION"/>
    <meta content="index,follow,noarchive" name="robots"/>
    <meta content="UCSD" name="SITE"/>
    <meta content="University of California San Diego" name="PAGETITLE"/>
    <meta content="The University California San Diego is one of the world's l
eading public research universities, located in beautiful La Jolla, Californi
a" name="DESCRIPTION"/>
    <!-- Facebook -->
    <meta content="University of California San Diego" property="og:title"/>
    <meta content="The University California San Diego is one of the world's l
eading public research universities, located in beautiful La Joll
```

Example: `POST` requests via `requests`

The following call to `requests.post` makes a post request to https://httpbin.org/post, with a `'name'` parameter of `'King Triton'`.

In [15]:
```
post_res = requests.post('https://httpbin.org/post',
                         data={'name': 'King Triton'})
post_res
```

Out[15]:
```
<Response [503]>
```

In [16]:
```
post_res.text
```

Out[16]:
```
'<html>\r\n<head><title>503 Service Temporarily Unavailable</title></head>\r
\n<body>\r\n<center><h1>503 Service Temporarily Unavailable</h1></center>\r\n
</body>\r\n</html>\r\n'
```

In [17]:
```
# More on this shortly!
post_res.json()
```

Loading [MathJax]/extensions/Safe.js

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
File ~/workbench/dsc80/.venv/lib/python3.12/site-packages/requests/models.py:9
74, in Response.json(self, **kwargs)
    973 try:
--> 974     return complexjson.loads(self.text, **kwargs)
    975 except JSONDecodeError as e:
    976     # Catch JSON-related errors and raise as requests.JSONDecodeError
    977     # This aliases json.JSONDecodeError and simplejson.JSONDecodeError

File ~/.local/share/uv/python/cpython-3.12.5-macos-aarch64-none/lib/python3.1
2/json/__init__.py:346, in loads(s, cls, object_hook, parse_float, parse_int,
parse_constant, object_pairs_hook, **kw)
    343 if (cls is None and object_hook is None and
    344         parse_int is None and parse_float is None and
    345         parse_constant is None and object_pairs_hook is None and not k
w):
--> 346     return _default_decoder.decode(s)
    347 if cls is None:

File ~/.local/share/uv/python/cpython-3.12.5-macos-aarch64-none/lib/python3.1
2/json/decoder.py:337, in JSONDecoder.decode(self, s, _w)
    333 """Return the Python representation of ``s`` (a ``str`` instance
    334 containing a JSON document).
    335
    336 """
--> 337 obj, end = self.raw_decode(s, idx=_w(s, 0).end())
    338 end = _w(s, end).end()

File ~/.local/share/uv/python/cpython-3.12.5-macos-aarch64-none/lib/python3.1
2/json/decoder.py:355, in JSONDecoder.raw_decode(self, s, idx)
    354 except StopIteration as err:
--> 355     raise JSONDecodeError("Expecting value", s, err.value) from None
    356 return obj, end

JSONDecodeError: Expecting value: line 1 column 1 (char 0)

During handling of the above exception, another exception occurred:

JSONDecodeError                           Traceback (most recent call last)
Cell In[17], line 2
      1 # More on this shortly!
----> 2 post_res.json()

File ~/workbench/dsc80/.venv/lib/python3.12/site-packages/requests/models.py:9
78, in Response.json(self, **kwargs)
    974     return complexjson.loads(self.text, **kwargs)
    975 except JSONDecodeError as e:
    976     # Catch JSON-related errors and raise as requests.JSONDecodeError
    977     # This aliases json.JSONDecodeError and simplejson.JSONDecodeError
--> 978     raise RequestsJSONDecodeError(e.msg, e.doc, e.pos)

JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

What happens when we try and make a `POST` request somewhere where we're unable to?

Loading [MathJax]/extensions/Safe.js

In [18]:
```
yt_res = requests.post('https://youtube.com',
                       data={'name': 'King Triton'})
yt_res
```

Out[18]:
```
<Response [400]>
```

In [19]:
```
yt_res.text
```

Loading [MathJax]/extensions/Safe.js

Out[19]:
'<html lang="en" dir="ltr"><head><title>Oops</title><style nonce="MOTuxGkryRm
ZjIRvJYn-PQ">html{font-family:Roboto,Arial,sans-serif;font-size:14px}body{bac
kground-color:#f9f9f9;margin:0}#content{max-width:440px;margin:128px auto}svg
{display:block;pointer-events:none}#monkey{width:280px;margin:0 auto}h1,p{tex
t-align:center;margin:0;color:#131313}h1{padding:24px 0 8px;font-size:24px;fo
nt-weight:400}p{line-height:21px}sentinel{}</style><link rel="shortcut icon"
href="https://www.youtube.com/img/favicon.ico" type="image/x-icon"><link rel
="icon" href="https://www.youtube.com/img/favicon_32.png" sizes="32x32"><link
rel="icon" href="https://www.youtube.com/img/favicon_48.png" sizes="48x48"><l
ink rel="icon" href="https://www.youtube.com/img/favicon_96.png" sizes="96x9
6"><link rel="icon" href="https://www.youtube.com/img/favicon_144.png" sizes
="144x144"></head><body> <h1>Something went wrong</h1> <sv
g id="monkey" viewBox="0 0 490 525"><path fill="#6A1B9A" d="M325 85c1 12-1 25
-5 38-8 29-31 52-60 61-26 8-54 14-81 18-37 6-26-37-38-72l-4-4c0-17-9-33 4-37l
33-4c9-2 9-21 11-30 1-7 3-14 5-21 8-28 40-42 68-29 18 9 36 19 50 32 13 11 16
31 17 48z"/><path fill="none" stroke="#6A1B9A" stroke-width="24" stroke-linec
ap="round" stroke-miterlimit="10" d="M431 232c3 15 21 19 34 11 15-9 14-30 5-4
3-12-17-38-25-59-10-23 18-27 53-21 97s1 92-63 108"/><path fill="#6A1B9A" d="M
284 158c35 40 63 85 86 133 24 52-6 113-62 123-2 0-4 1-6 1-53 9-101-33-101-87V
188l83-30z"/><path fill="#F7CB4D" d="M95 152c-3-24 13-57 46-64l27-5c9-2 16-19
17-28l3-15 20-3c44 14 42 55 18 69 22 0 39 26 32 53-5 18-20 32-39 36-13 3-26 5
-40 8-50 8-80-14-84-51z"/><path fill="#6A1B9A" d="M367 392c-21 18-77 70-25 11
9h-61c-27-29-32-69 1-111l85-8z"/><path fill="#6A1B9A" d="M289 399c-21 18-84 6
2-32 111h-61c-37-34-5-104 43-134l50 23z"/><path fill="#EDB526" d="M185 56l3-1
5 20-3c25 8 35 25 35 41-12-18-49-29-58-23z"/><path fill="#E62117" d="M190 34c
8-28 40-42 68-29 18 9 36 19 50 32 10 9 14 23 16 37L187 46l3-12z"/><path fill
="#8E24AA" d="M292 168c0 0 0 201 0 241s20 98 91 85l-16-54c-22 12-31-17-31-37
0-20 0-108 0-137S325 200 292 168z"/><path fill="#F7CB4D" d="M284 79c11-9 23-1
7 35-23 25-12 54 7 59 38v1c4 27-13 51-36 53-12 1-25 1-37 0-22-1-39-27-32-52v-
1c2-6 6-12 11-16z"/><path fill="#6A1B9A" d="M201 203s0 84-95 140l22 42s67-25
89-86-16-96-16-96z"/><path fill="#BE2117" d="M224 54l-67-14c-10-2-13-15-5-21s
18-6 26 0l46 35z"/><circle fill="#4A148C" cx="129" cy="161" r="12"/><circle f
ill="#4A148C" cx="212" cy="83" r="7"/><circle fill="#4A148C" cx="189" cy="79"
r="7"/><path fill="#F7CB4D" d="M383 493c11-3 19-8 25-13 7-10 4-16-5-20 8-9 2-
22-8-18 1-1 1-2 1-3 3-9-9-15-15-8-3 4-8 7-13 9l15 53z"/><path fill="#EDB526"
d="M252 510c5 6 0 15-9 15h-87c-10 0-16-8-13-15 5-12 21-19 36-16l73 16z"/><ell
ipse transform="rotate(19.126 278.35 14.787)" fill="#E62117" cx="278" cy="15"
rx="9" ry="7"/><path fill="#F7CB4D" d="M341 510c5 6 0 15-9 15h-87c-10 0-16-8-
13-15 5-12 21-19 36-16l73 16z"/><path fill="#EDB526" d="M357 90c-12-19-35-23-
55-11-19 12-25 32-13 52"/><path fill="#E62117" d="M110 427l21-9c5-2 7-8 5-13l
-42-94c-3-6-9-9-15-6l-11 5c-6 2-9 9-7 15l36 97c2 5 8 7 13 5z"/><path fill="#B
0BEC5" d="M37 278l41-17c11-4 22-5 33-1 5 2 10 4 14 6 6 3 4 11-3 11-9 0-18 1-2
6 3l2 12c1 6-2 11-8 13l-36 15c-5 2-10 1-14-2l-9-7-2 17c0 2-2 4-4 5l-3 1c-3 1-
7 0-8-3L1 300c-1-3 0-7 4-9l4-2c2-1 5 0 7 1l12 10 1-11c0-5 3-9 8-11z"/><path f
ill="#F7CB4D" d="M103 373c10 2 14 10 8 19 6-1 10 4 10 9 0 3-3 6-6 7l-26 11c-2
1-5 1-8 0-6-3-7-9-2-16-7-1-13-9-6-17-8-1-12-8-8-15l3-3 23-11c9-4 19 8 12 16
z"/><ellipse transform="rotate(173.3 233.455 334.51)" fill="#8E24AA" cx="234"
cy="335" rx="32" ry="46"/></svg> <style nonce="MOTuxGkryRmZjIRvJYn-PQ">#yt
-masthead{margin:15px auto;width:440px;margin-top:25px}#logo-container{margin
-right:5px;float:left;cursor:pointer;text-decoration:none}.logo{background:ce
nter/contain no-repeat url(//www.gstatic.com/youtube/img/branding/youtubelog
o/2x/youtubelogo_50.png);width:128px;height:30px;cursor:pointer;display:inlin
e-block}#masthead-search{display:-webkit-box;display:-webkit-flex;display:fle
x;margin-top:3px;max-width:650px;overflow:hidden;padding:0;position:relativ
ton{border-left:0;border-top-left-radius:0;border-bottom-left-ra
dius:0;float:right;height:29px;padding:0;border:solid 1px transparent;border-

Loading [MathJax]/extensions/Safe.js

color:#d3d3d3;background:#f8f8f8;color:#333;cursor:pointer}.search-button:hover{border-color:#c6c6c6;background:#f0f0f0;-webkit-box-shadow:0 1px 0 rgba(0,0,0,.1);box-shadow:0 1px 0 rgba(0,0,0,.1)}.search-button-content{border:none;display:block;opacity:.6;padding:0;text-indent:-10000px;background:no-repeat url(//www.gstatic.com/youtube/src/web/htdocs/img/search.png);-webkit-background-size:auto auto;background-size:auto;width:15px;height:15px;-webkit-box-shadow:none;box-shadow:none;margin:0 25px}#masthead-search-terms-border{-webkit-box-flex:1;-webkit-flex:1 1 auto;flex:1 1 auto;border:1px solid #ccc;-webkit-box-shadow:inset 0 1px 2px #eee;box-shadow:inset 0 1px 2px #eee;background-color:#fff;font-size:14px;height:29px;line-height:30px;margin:0 0 2px;overflow:hidden;position:relative;-webkit-box-sizing:border-box;box-sizing:border-box;-webkit-transition:border-color .2s ease;transition:border-color .2s ease}#masthead-search-terms{background:transparent;border:0;font-size:16px;height:100%;left:0;margin:0;outline:none;padding:2px 6px;position:absolute;width:100%;-webkit-box-sizing:border-box;box-sizing:border-box}sentinel{}</style> <form id="masthead-search" class="search-form" action="https://www.youtube.com/results"><script nonce="2svJ7G9970MZ3341H1-5TQ">document.addEventListener(\'DOMContentLoaded\', function () {document.getElementById(\'masthead-search\').addEventListener(\'submit\', function(e) {if (document.getElementById(\'masthead-search-terms\').value == \'\') {e.preventDefault();}});});</script> <input id="masthead-search-terms" autocomplete="off" name="search_query" value="" type="text" placeholder="Search" title="Search" aria-label="Search"><script nonce="2svJ7G9970MZ3341H1-5TQ">document.addEventListener(\'DOMContentLoaded\', function () {document.getElementById(\'masthead-search-terms\').addEventListener(\'keydown\', function() {if (!this.value && (event.keyCode == 40 || event.keyCode == 32 || event.keyCode == 34)) {this.onkeydown = null; this.blur();}});});</script> <button id="masthead-search-button" class="search-button" type="submit" dir="ltr"><script nonce="2svJ7G9970MZ3341H1-5TQ">document.addEventListener(\'DOMContentLoaded\', function () {document.getElementById(\'masthead-search-button\').addEventListener(\'click\', function(e) {if (document.getElementById(\'masthead-search-terms\').value == \'\') {e.preventDefault(); return;}e.preventDefault(); document.getElementById(\'masthead-search\').submit();});});</script>Search</button></form> </body></html>'

`yt_res.text` is a string containing HTML – we can render this in-line using `IPython.display.HTML`.

In [20]:
```
from IPython.display import HTML
```

In [21]:
```
HTML(yt_res.text)
```

Loading [MathJax]/extensions/Safe.js

Out[21]:

HTTP status codes

- When we **request** data from a website, the server includes an **HTTP status code** in the response.
- The most common status code is `200`, which means there were no issues.
- Other times, you will see a different status code, describing some sort of event or

Loading [MathJax]/extensions/Safe.js

  - Common examples: `400` – bad request, `404` – page not found, `500` – internal server error.
  - The first digit of a status describes its general "category".

- See https://httpstat.us for a list of all HTTP status codes.
  - It also has example sites for each status code; for example, https://httpstat.us/404 returns a `404`.

In [22]:
```
yt_res.status_code
```

Out[22]:
```
400
```

In [23]:
```
# ok checks if the result was successful.
yt_res.ok
```

Out[23]:
```
False
```

Handling unsuccessful requests

- Unsuccessful requests can be re-tried, depending on the issue.
  - A good first step is to wait a little, then try again.
- A common issue is that you're making too many requests to a particular server at a time – if this is the case, increase the time between each request. You can even do this programatically, say, using `time.sleep`.
- See the textbook for more examples.

In [ ]:
```

```

Data formats

The data formats of the internet

Responses typically come in one of two formats: HTML or JSON.

- The response body of a `GET` request is usually either JSON (when using an API) or HTML (when accessing a webpage).
- The response body of a `POST` request is usually JSON.
- XML is also a common format, but not as popular as it once was.

JSON

Loading [MathJax]/extensions/Safe.js

- JSON stands for **JavaScript Object Notation**. It is a lightweight format for storing and transferring data.
- It is:
  - very easy for computers to read and write.
  - moderately easy for programmers to read and write by hand.
  - meant to be generated and parsed.
- Most modern languages have an interface for working with JSON objects.
  - JSON objects *resemble* Python dictionaries (but are not the same!).

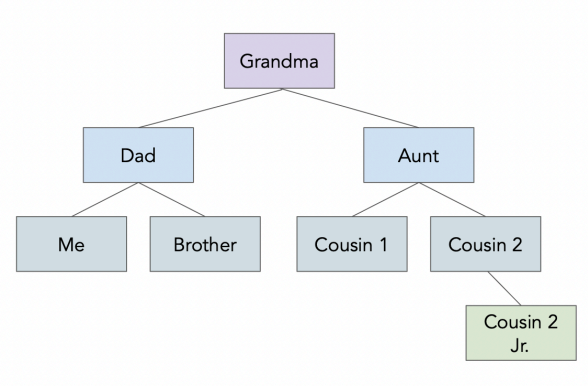
JSON data types

Type	Description
String	Anything inside double quotes.
Number	Any number (no difference between ints and floats).
Boolean	`true` and `false`.
Null	JSON's empty value, denoted by `null`.
Array	Like Python lists.
Object	A collection of key-value pairs, like dictionaries. Keys must be strings, values can be anything (even other objects).

See json-schema.org for more details.

Example JSON object

See `data/family.json`.



In [24]:
```
import json
from pathlib import Path

f = Path('data') / 'family.json'
json.loads(f.read_text())
```

Loading [MathJax]/extensions/Safe.js

In [25]:
```
family_tree
```

Out[25]:
```
{'name': 'Grandma',
 'age': 94,
 'children': [{'name': 'Dad',
   'age': 60,
   'children': [{'name': 'Me', 'age': 24}, {'name': 'Brother', 'age': 22}]},
  {'name': 'My Aunt',
   'children': [{'name': 'Cousin 1', 'age': 34},
    {'name': 'Cousin 2',
     'age': 36,
     'children': [{'name': 'Cousin 2 Jr.', 'age': 2}]}]}]}
```

In [26]:
```
family_tree['children'][1]['children'][0]['age']
```

Out[26]:
```
34
```

Aside: `eval`

- `eval`, which stands for "evaluate", is a function built into Python.
- It takes in a **string containing a Python expression** and evaluates it in the current context.

In [27]:
```
x = 4
eval('x + 5')
```

Out[27]:
```
9
```

- It seems like `eval` can do the same thing that `json.loads` does...

In [28]:
```
eval(f.read_text())
```

Out[28]:
```
{'name': 'Grandma',
 'age': 94,
 'children': [{'name': 'Dad',
   'age': 60,
   'children': [{'name': 'Me', 'age': 24}, {'name': 'Brother', 'age': 22}]},
  {'name': 'My Aunt',
   'children': [{'name': 'Cousin 1', 'age': 34},
    {'name': 'Cousin 2',
     'age': 36,
     'children': [{'name': 'Cousin 2 Jr.', 'age': 2}]}]}]}
```

- But you should **almost never use `eval`**. The next slide demonstrates why.

`eval` gone wrong

Loading [MathJax]/extensions/Safe.js appens when we use `eval` on a string representation of a JSON object:

In [29]:
```
f_other = Path('data') / 'evil_family.json'
eval(f_other.read_text())
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[29], line 2
      1 f_other = Path('data') / 'evil_family.json'
----> 2 eval(f_other.read_text())

File <string>:6

Cell In[1], line 5, in err()
      4 def err():
----> 5     raise ValueError('i just deleted all your files lol 😂')

ValueError: i just deleted all your files lol 😂
```

- Oh no! Since `evil_family.json`, which could have been downloaded from the internet, contained malicious code, we now lost all of our files.
- This happened because `eval` **evaluates** all parts of the input string as if it were Python code.
- You never need to do this – instead, use the `.json()` method of a response object, or use the `json` library.

Using the `json` module

Let's process the same file using the `json` module. Note:

- `json.load(f)` loads a JSON file from a file object.
- `json.loads(f)` loads a JSON file from a **s**tring.

In [30]:
```
f_other = Path('data') / 'evil_family.json'
s = f_other.read_text()
s
```

Out[30]:
```
'{\n    "name": "Grandma",\n    "age": 94,\n    "children": [\n        {\n
"name": err(),\n        "age": 60,\n        "children": [{"name": "Me", "ag
e": 24}, \n                    {"name": "Brother", "age": 22}]\n        },\n
{\n        "name": "My Aunt",\n        "children": [{"name": "Cousin 1", "ag
e": 34}, \n                    {"name": "Cousin 2", "age": 36, "children":
\n                        [{"name": "Cousin 2 Jr.", "age": 2}]\n
}\n                    ]\n        }\n    ]\n}'
```

In [31]:
```
json.loads(s)
```

Loading [MathJax]/extensions/Safe.js

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
Cell In[31], line 1
----> 1 json.loads(s)

File ~/.local/share/uv/python/cpython-3.12.5-macos-aarch64-none/lib/python3.12/json/__init__.py:346, in loads(s, cls, object_hook, parse_float, parse_int, parse_constant, object_pairs_hook, **kw)
    341     s = s.decode(detect_encoding(s), 'surrogatepass')
    343 if (cls is None and object_hook is None and
    344         parse_int is None and parse_float is None and
    345         parse_constant is None and object_pairs_hook is None and not kw):
--> 346     return _default_decoder.decode(s)
    347 if cls is None:
    348     cls = JSONDecoder

File ~/.local/share/uv/python/cpython-3.12.5-macos-aarch64-none/lib/python3.12/json/decoder.py:337, in JSONDecoder.decode(self, s, _w)
    332 def decode(self, s, _w=WHITESPACE.match):
    333     """Return the Python representation of ``s`` (a ``str`` instance
    334     containing a JSON document).
    335
    336     """
--> 337     obj, end = self.raw_decode(s, idx=_w(s, 0).end())
    338     end = _w(s, end).end()
    339     if end != len(s):

File ~/.local/share/uv/python/cpython-3.12.5-macos-aarch64-none/lib/python3.12/json/decoder.py:355, in JSONDecoder.raw_decode(self, s, idx)
    353     obj, end = self.scan_once(s, idx)
    354 except StopIteration as err:
--> 355     raise JSONDecodeError("Expecting value", s, err.value) from None
    356 return obj, end

JSONDecodeError: Expecting value: line 6 column 17 (char 84)
```

- Since `util.err()` is not a string in JSON (there are no quotes around it), `json.loads` is not able to parse it as a JSON object.
- This "safety check" is intentional.

Handling *unfamiliar* data

- Never trust data from an unfamiliar site.
- **Never** use `eval` on "raw" data that you didn't create!
- The JSON data format needs to be **parsed**, not evaluated as a dictionary.
  - It was designed with safety in mind!

Loading [MathJax]/extensions/Safe.js

Aside: `pd.read_json`

`pandas` also has a built-in `read_json` function.

In [32]:
```
pd.read_json(f)
```

Out[32]:

	name	age	children
0	Grandma	94	{'name': 'Dad', 'age': 60, 'children': [{'name...
1	Grandma	94	{'name': 'My Aunt', 'children': [{'name': 'Cou...

It only makes sense to use it, though, when you have a JSON file that has some sort of tabular structure. Our family tree example does not.

APIs and scraping

Programmatic requests

- We learned how to use the Python `requests` package to exchange data via HTTP.
  - `GET` requests are used to request data **from** a server.
  - `POST` requests are used to **send** data to a server.
- There are two ways of collecting data through a request:
  - By using a published API (application programming interface).
  - By scraping a webpage to collect its HTML source code.

APIs

An application programming interface (API) is a service that makes data directly available to the user in a convenient fashion.

Advantages:

- The data are usually clean, up-to-date, and ready to use.
- The presence of a API signals that the data provider is okay with you using their data.
- The data provider can plan and regulate data usage.
  - Some APIs require you to create an API "key", which is like an account for using the API.
  - APIs can also give you access to data that isn't publicly available on a webpage.

Big disadvantage: APIs don't always exist for the data you want!

Loading [MathJax]/extensions/Safe.js

API terminology

- A URL, or uniform resource locator, describes the location of a website or resource.
- An **API endpoint** is a URL of the data source that the user wants to make requests to.
- For example, on the Reddit API:
  - the `/comments` endpoint retrieves information about comments.
  - the `/hot` endpoint retrieves data about posts labeled "hot" right now.
  - To access these endpoints, you add the endpoint name to the base URL of the API.

API requests

- API requests are just `GET` / `POST` requests to a specially maintained URL.
- Let's test out the Pokémon API.

First, let's make a `GET` request for `'squirtle'`. To do this, we need to make a request to the correct URL.

In [33]:
```
def create_url(pokemon):
    return f'https://pokeapi.co/api/v2/pokemon/{pokemon}'

create_url('squirtle')
```

Out[33]:
```
'https://pokeapi.co/api/v2/pokemon/squirtle'
```

In [34]:
```
r = requests.get(create_url('squirtle'))
r
```

Out[34]:
```
<Response [200]>
```

Remember, the 200 status code is good! Let's take a look at the **content**:

In [35]:
```
r.content[:1000]
```

Loading [MathJax]/extensions/Safe.js

Out[35]:
b'{"abilities":[{"ability":{"name":"torrent","url":"https://pokeapi.co/api/v2/ability/67/"},"is_hidden":false,"slot":1},{"ability":{"name":"rain-dish","url":"https://pokeapi.co/api/v2/ability/44/"},"is_hidden":true,"slot":3}],"base_experience":63,"cries":{"latest":"https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon/latest/7.ogg","legacy":"https://raw.githubusercontent.com/PokeAPI/cries/main/cries/pokemon/legacy/7.ogg"},"forms":[{"name":"squirtle","url":"https://pokeapi.co/api/v2/pokemon-form/7/"}],"game_indices":[{"game_index":177,"version":{"name":"red","url":"https://pokeapi.co/api/v2/version/1/"}},{"game_index":177,"version":{"name":"blue","url":"https://pokeapi.co/api/v2/version/2/"}},{"game_index":177,"version":{"name":"yellow","url":"https://pokeapi.co/api/v2/version/3/"}},{"game_index":7,"version":{"name":"gold","url":"https://pokeapi.co/api/v2/version/4/"}},{"game_index":7,"version":{"name":"silver","url":"https://pokeapi.co/api/v2/version/5/"}},{"game_index":7,"v'

Looks like JSON. We can extract the JSON from this request with the `json` method (or by passing `r.text` to `json.loads`).

In [36]:
```
rr = r.json()
rr.keys()
```

Out[36]:
```
dict_keys(['abilities', 'base_experience', 'cries', 'forms', 'game_indices', 'height', 'held_items', 'id', 'is_default', 'location_area_encounters', 'moves', 'name', 'order', 'past_abilities', 'past_types', 'species', 'sprites', 'stats', 'types', 'weight'])
```

In [37]:
```
rr['weight']
```

Out[37]:
```
90
```

In [38]:
```
rr['abilities'][1]['ability']['name']
```

Out[38]:
```
'rain-dish'
```

Let's try a `GET` request for `'billy'`.

In [39]:
```
r = requests.get(create_url('billy'))
r
```

Out[39]:
```
<Response [404]>
```

We receive a 404 error, since there is no Pokemon named `'billy'`!

Scraping

Scraping is the act of programmatically "browsing" the web, downloading the source code (HTML) of pages that you're interested in extracting data from.

Big advantage: You can always do it! For example, Google scrapes webpages in order to make them searchable.

Loading [MathJax]/extensions/Safe.js

- It is often difficult to parse and clean scraped data.
  - Source code often includes a lot of content unrelated to the data you're trying to find (e.g. formatting, advertisements, other text).
- Websites can change often, so scraping code can get outdated quickly.
- Websites may not want you to scrape their data!
- **In general, we prefer APIs, but scraping is a useful skill to learn.**

Example: Scraping the HDSI faculty page

To fully understand how to scrape, we need to understand how HTML documents are structured and how to extract information out of them.

But as a preview of what's to come next week, let's start by making a request to the HDSI Faculty page, https://datascience.ucsd.edu/faculty.

In [40]:
```
import certifi
certifi.where()
```

Out[40]:
```
'/Users/eldridge/workbench/dsc80/.venv/lib/python3.12/site-packages/certifi/cacert.pem'
```

In [41]:
```
# Sometimes, the requests library gets weirdly strict about the HDSI webpage,
# so we'll skip its security checks using verify=False.
fac_response = requests.get('https://datascience.ucsd.edu/faculty/', verify=Fa
fac_response
```

```
/Users/eldridge/workbench/dsc80/.venv/lib/python3.12/site-packages/urllib3/connectionpool.py:1099: InsecureRequestWarning:

Unverified HTTPS request is being made to host 'datascience.ucsd.edu'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
```

Out[41]:
```
<Response [200]>
```

The response is a long HTML document.

In [42]:
```
len(fac_response.text)
```

Out[42]:
```
258316
```

In [43]:
```
print(fac_response.text[:1000])
```

Loading [MathJax]/extensions/Safe.js

```
<!DOCTYPE html><html lang="en-US"><head><meta charset="UTF-8"><meta name="view
port" 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 &#8211; UC San
Diego</title> <script defer src="data:text/javascript;base64,CiAgICAgICAgICAgI
CAgICAgICAgICAgICAgICAvKiBZb3UgY2FuIGFkZCBtb3JlIGNvbmZpZ3VyYXRpb24gb3B0aW9ucyB
0byB3ZWJmb250bG9hZGVyIGJ5IHByZXZpb3VzbHkgZGVmaW5pbmcgdGhlIFdlYkZvbnRDb25maWcgd
2l0aCB5b3VyIG9wdGlvbnMgKi8KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIGlmICggdHlwZW9
mIFdlYkZvbnRDb25maWcgPT09ICJ1bmRlZmluZWQiICkgewogICAgICAgICAgICAgICAgICAgICAgICAgI
CAgICAgICAgIFdlYkZvbnRDb25maWcgPSBuZXcgT2JqZWN0KCk7CiAgICAgICAgICAgICAgICAgICAgICA
gICAgICAgICB9CiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICBXZWJGb250Q29uZmlnWydnb29nb
GUnXSA9IHtmYW1pbGllczogWydSb2JvdG86NDAwLDUwMCcsICdKb3N0OjQwMCw3MDAnLCAnVGVrbzo
3MDAnXX07CgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgKGZ1bmN0aW9uKCk
```


To **parse** HTML, we'll use the BeautifulSoup library.

In [44]:
```python
from bs4 import BeautifulSoup
soup = BeautifulSoup(fac_response.text)
```

Now, `soup` is a representation of the faculty page's HTML code that Python knows how to extract information from.

In [45]:
```python
# Magic that we'll learn how to create together next Tuesday.
divs = soup.find_all('div', class_='vc_grid-item')
names = [div.find('h4').text for div in divs]
titles = [div.find(class_='pendari_people_title').text for div in divs]

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

Out[45]:

	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 [46]:
```python
faculty[faculty['title'].str.contains('Lecturer') | faculty['title'].str.conta
```

Loading [MathJax]/extensions/Safe.js

Out[46]:

	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 [47]:
```
from IPython.display import Image, display

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

In [48]:
```
show_picture('justin')
```

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.

Loading [MathJax]/extensions/Safe.js

Consequences of irresponsible scraping

If you make too many requests:

- The server may block your IP Address.
- You may take down the website.
  - 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 [49]:
```
!cat data/lec10_ex1.html
```

Loading [MathJax]/extensions/Safe.js

```
<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 [50]:
```
from IPython.display import HTML
HTML(filename=Path('data') / 'lec10_ex1.html')
```

Out[50]:

This is a heading

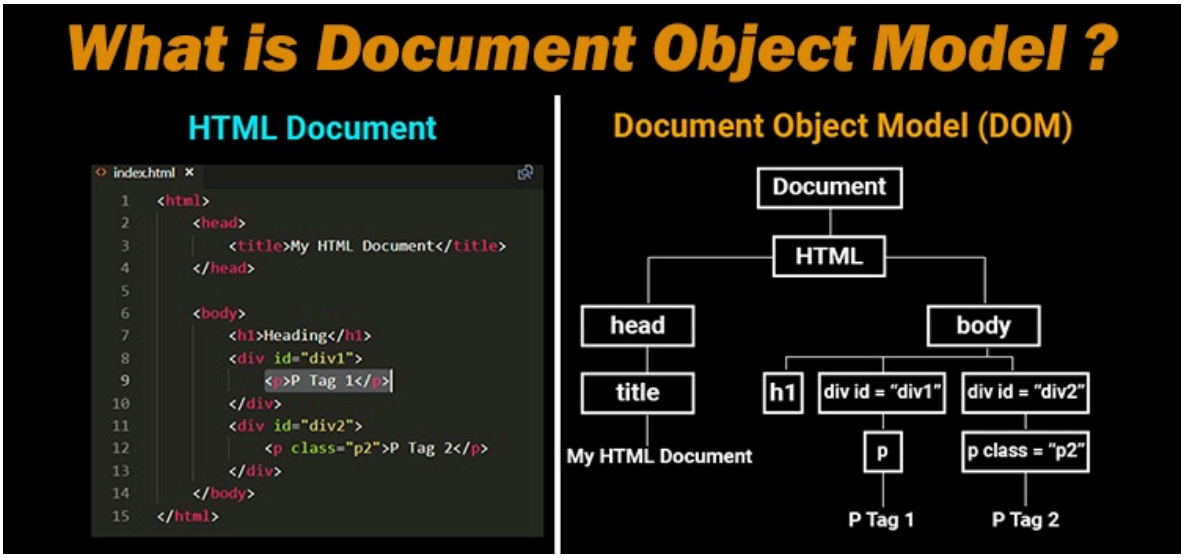
This is a paragraph.

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>`.

Loading [MathJax]/extensions/Safe.js



(source)

Useful tags to know

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

Loading [MathJax]/extensions/Safe.js

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

What do you think this webpage looks like?

In [51]:
```
!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/boots
trap.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

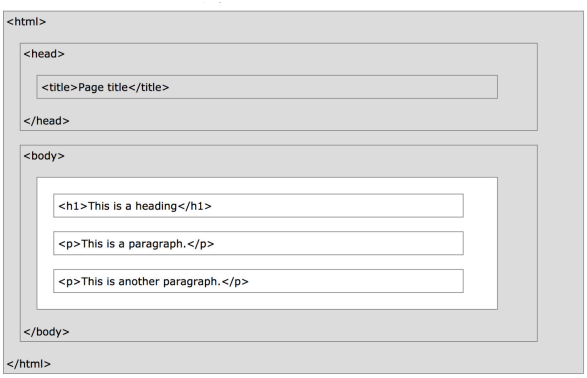
```
<div style="background-color:lightblue">
  <h3>This is a heading</h3>
  <p>This is a paragraph.</p>
</div>
```

- 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**!

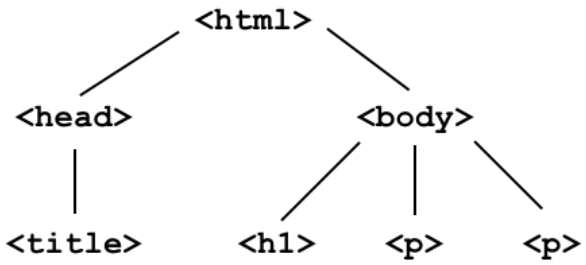
Document trees

Loading [MathJax]/extensions/Safe.js

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?



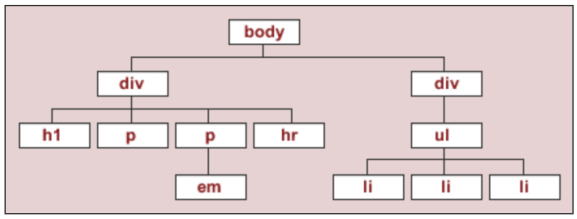
Parsing HTML using Beautiful Soup

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".

Loading [MathJax]/extensions/Safe.js

In [52]:
```
html_string = '''
<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>
'''.strip()
```

In [53]:
```
HTML(html_string)
```

Out[53]:

Heading here

My First paragraph

My *second* paragraph

- item 1
- item 2
- item 3

`BeautifulSoup` objects

Loading [MathJax]/extensions/Safe.js

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

In [54]:
```
import bs4
```

In [55]:
```
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
```

Loading [MathJax]/extensions/Safe.js

```
 `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/__ini
t__.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 [56]:
```
soup = bs4.BeautifulSoup(html_string)
soup
```

Loading [MathJax]/extensions/Safe.js

Out[56]:
```
<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 [57]:
```
type(soup)
```

Out[57]:
```
bs4.BeautifulSoup
```

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

In [58]:
```
print(soup.text)
```

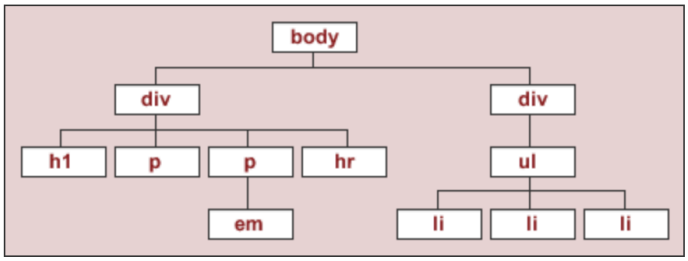
```
Heading here
My First paragraph
My second paragraph



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.

Loading [MathJax]/extensions/Safe.js



In [59]:
```
soup.descendants
```

Out[59]:
```
<generator object Tag.descendants at 0x114bdc5f0>
```

In [60]:
```
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:

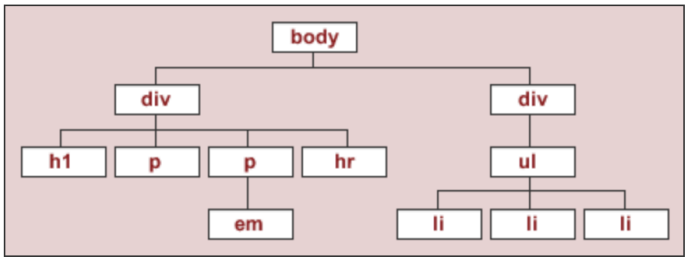
- `soup.find(tag)`, which finds the **first** instance of a tag (the first one on the page, i.e. the first one that DFS sees).
  - More general: `soup.find(name=None, attrs={}, recursive=True, text=None, **kwargs)`.
- `soup.find_all(tag)` will find **all** instances of a tag.

`find` finds tags!

Using `find`

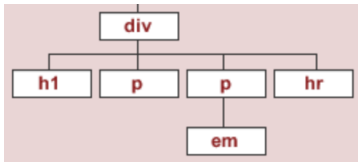
Let's try and extract the first `<div>` subtree.

Loading [MathJax]/extensions/Safe.js



In [61]:
```
soup.find('div')
```

Out[61]:
```
<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 [62]:
```
soup.find('div', attrs={'id': 'nav'})
```

Out[62]:
```
<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 [63]:
```
# The ul child is not at the top of the tree, but we can still find it.
soup.find('ul')
```

Out[63]:
```
<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 [64]:
```
soup.find_all('div')
```

Loading [MathJax]/extensions/Safe.js

Out[64]:
```
[<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 [65]:
```
soup.find_all('li')
```

Out[65]:
```
[<li>item 1</li>, <li>item 2</li>, <li>item 3</li>]
```

In [66]:
```
[x.text for x in soup.find_all('li')]
```

Out[66]:
```
['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 [67]:
```
soup.find('p')
```

Out[67]:
```
<p>My First paragraph</p>
```

In [68]:
```
soup.find('p').text
```

Out[68]:
```
'My First paragraph'
```

In [69]:
```
soup.find('div')
```

Out[69]:
```
<div id="content">
<h1>Heading here</h1>
<p>My First paragraph</p>
<p>My <em>second</em> paragraph</p>
<hr/>
</div>
```

In [70]:
```
soup.find('div').text
```

Out[70]:
```
'\nHeading here\nMy First paragraph\nMy second paragraph\n\n'
```

In [71]:
```
soup.find('div').attrs
```

Loading [MathJax]/extensions/Safe.js

Out[71]:
```
{'id': 'content'}
```

In [72]:
```
soup.find('div').get('id')
```

Out[72]:
```
'content'
```

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

In [73]:
```
soup
```

Out[73]:
```
<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 [74]:
```
# While there are multiple 'id' attributes, none of them are in the <html> tag
soup.get('id')
```

In [75]:
```
soup.find('div').get('id')
```

Out[75]:
```
'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>
```

Loading [MathJax]/extensions/Safe.js

```
            </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:

Loading [MathJax]/extensions/Safe.js

	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,miracle,

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.

Loading [MathJax]/extensions/Safe.js

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 [76]:
```
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 [77]:
```
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 [78]:
```
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'})
```

Loading [MathJax]/extensions/Safe.js

In [79]:
```
divs[0]
```

Out[79]:
```
<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
<span class="text" itemprop="text">“The world as we have created it is a proc
ess of our thinking. It cannot be changed without changing our thinking.”</sp
an>
<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,wor
ld" 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 [80]:
```
divs[0]
```

Out[80]:
```
<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
<span class="text" itemprop="text">“The world as we have created it is a proc
ess of our thinking. It cannot be changed without changing our thinking.”</sp
an>
<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,wor
ld" 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 [81]:
```
# The quote.
divs[0].find('span', class_='text').text
```

Out[81]:
```
'“The world as we have created it is a process of our thinking. It cannot be
changed without changing our thinking.”'
```

In [82]:
```
# The author.
divs[0].find('small', class_='author').text
```

Out[82]:
```
'Albert Einstein'
```

In [83]:
```
# The URL for the author.
divs[0].find('a').get('href')
```

Loading [MathJax]/extensions/Safe.js

Out[83]:
```
'/author/Albert-Einstein'
```

In [84]:
```
# The quote's tags.
divs[0].find('meta', class_='keywords').get('content')
```

Out[84]:
```
'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 [85]:
```
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, 'tags'
```

In [86]:
```
process_quote(divs[-1])
```

Out[86]:
```
{'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 [87]:
```
def process_page(divs):
    return pd.DataFrame([process_quote(div) for div in divs])
```

In [88]:
```
process_page(divs)
```

Loading [MathJax]/extensions/Safe.js

Out[88]:

	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-thoughts,t
1	“It is our choices, Harry, that show what we t...	J.K. Rowling	https://quotes.toscrape.com/author/J-K-Rowling	ab
2	“There are only two ways to live your life. On...	Albert Einstein	https://quotes.toscrape.com/author/Albert-Eins...	inspirational,life,live,mi
...	...	...	...	
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,inspirational
8	“A woman is like a tea bag; you never know how...	Eleanor Roosevelt	https://quotes.toscrape.com/author/Eleanor-Roo...	misattributed-elea
9	“A day without sunshine is like, you know, nig...	Steve Martin	https://quotes.toscrape.com/author/Steve-Martin	humor,c

10 rows × 4 columns

Putting it all together

Loading [MathJax]/extensions/Safe.js

In [89]:
```
def make_quote_df(n):
    '''Returns a DataFrame containing the quotes on the first n pages of https
    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 [90]:
```
quotes = make_quote_df(3)
quotes.head()
```

Out[90]:

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

In [91]:
```
quotes[quotes['author'] == 'Albert Einstein']
```

Loading [MathJax]/extensions/Safe.js

Out[91]:

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

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

Loading [MathJax]/extensions/Safe.js

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 [92]:
```
fac_response = requests.get('https://datascience.ucsd.edu/faculty/', verify=Fa
fac_response
```

```
/Users/eldridge/workbench/dsc80/.venv/lib/python3.12/site-packages/urllib3/con
nectionpool.py:1099: InsecureRequestWarning:

Unverified HTTPS request is being made to host 'datascience.ucsd.edu'. Adding
certificate verification is strongly advised. See: https://urllib3.readthedoc
s.io/en/latest/advanced-usage.html#tls-warnings
```

Out[92]:
```
<Response [200]>
```

In [93]:
```
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 [94]:
```
divs = soup.find_all(
    class_='vc_grid-item',
)
```

In [95]:
```
len(divs)
```

Out[95]:
```
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 [96]:
```
divs[0]
```

Loading [MathJax]/extensions/Safe.js

Out[96]:
```
<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-altintas"></a><div class="vc_grid-item-mini vc_clearfix"><div class="vc_gitem-animated-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_headshot.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.edu/wp-content/uploads/2022/10/ilkayaltintas_headshot.jpg"/><div class="vc_gitem-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-source-post_title"><h4 style="text-align: left"><a href="https://datascience.ucsd.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 class="excerpt"></div><div class="terms">Faculty Fellows Council Faculty</div></div></div></div></div></div></div>
```

In [97]:
```
divs[0].find('h4').text
```

Out[97]:
```
'Ilkay Altintas'
```

We can also extract job titles:

In [98]:
```
divs[0].find(class_='field').text
```

Out[98]:
```
'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 [99]:
```
names = [div.find('h4').text for div in divs]
names[:10]
```

Out[99]:
```
['Ilkay Altintas',
 'Tiffany Amariuta',
 'Mikio Aoi',
 'Ery Arias-Castro',
 'Vineet Bafna',
 'Mikhail Belkin',
 'Umesh Bellur',
 'Peter Chi',
 'Henrik Christensen',
 'Alex Cloninger']
```

In [100…
```
titles = [div.find(class_='field').text for div in divs]
titles[:10]
```

Loading [MathJax]/extensions/Safe.js

Out[100…
```
['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 [101…
```
faculty = pd.DataFrame({
    'name': names,
    'title': titles,
})
faculty.head()
```

Out[101…

	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 [102…
```
faculty[faculty['title'].str.contains('Teaching') | faculty['title'].str.conta
```

Out[102…

	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 [103…
```
from IPython.display import Image, display

ure(name):
```

Loading [MathJax]/extensions/Safe.js

```
    idx = faculty[faculty['name'].str.lower().str.contains(name.lower())].inde
    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>
```

Loading [MathJax]/extensions/Safe.js

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 [illegible] diffused to more than 1000 people. Falsehood also diffused faster than

Loading [MathJax]/extensions/Safe.js

> 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.

In [ ]:

Loading [MathJax]/extensions/Safe.js