LECTURE 37

PARSING AND SCRAPING HTML

MCS 275 Spring 2023

David Dumas

LECTURE 37: PARSING AND SCRAPING HTML

Reminders and announcements:

- Project 4 is due 6pm CDT Friday 28 April.
- Please install `beautifulsoup4` with

```
python3 -m pip install beautifulsoup4
```

- I added a demo program that shows how to generate and serve an image in Flask (without writing it to a file).

GETTING DATA FROM THE WEB

APIs that directly serve machine-readable, typed data are the best way to bring data from an external service into your programs.

Extracting data from HTML — a language for making human-readable documents — should be considered a last resort.

TODAY

We discuss what you can do if:

- There is no API, but there is HTML containing the data you need, or
- The structure of an HTML document **is** the data.

SIMPLE HTML PROCESSING

Level 0: Treat HTML as a string. Do string things.

Level 1: Treat HTML as a stream of tags, attributes, and text. Have a HTML parser recognize them and tell you what it finds. `html.parser` is good for this.

These approaches handle huge documents efficiently, but make nontrivial data extraction quite complex.

HTML DOCUMENT AS AN OBJECT

Level 2: Use a higher-level HTML data extraction framework like Beautiful Soup, Scrapy, or Selenium.

These frameworks create a data structure that represents the entire document, supporting various kinds of searching, traversal, and extraction.

Note that the whole document needs to fit in memory.

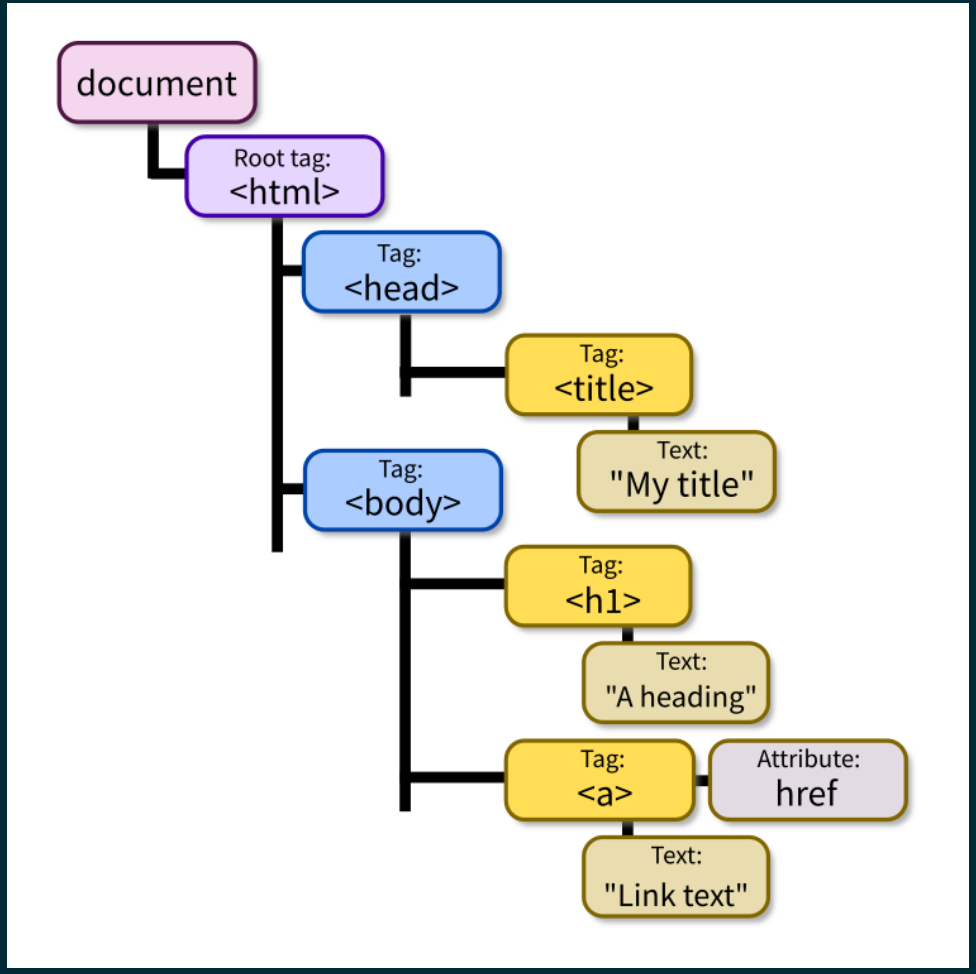
DOM

The **Document Object Model** or DOM is a language-independent model for representing a HTML document as a tree of nodes.

Each node represents part of the document, such as a tag, an attribute, or text appearing inside a tag.

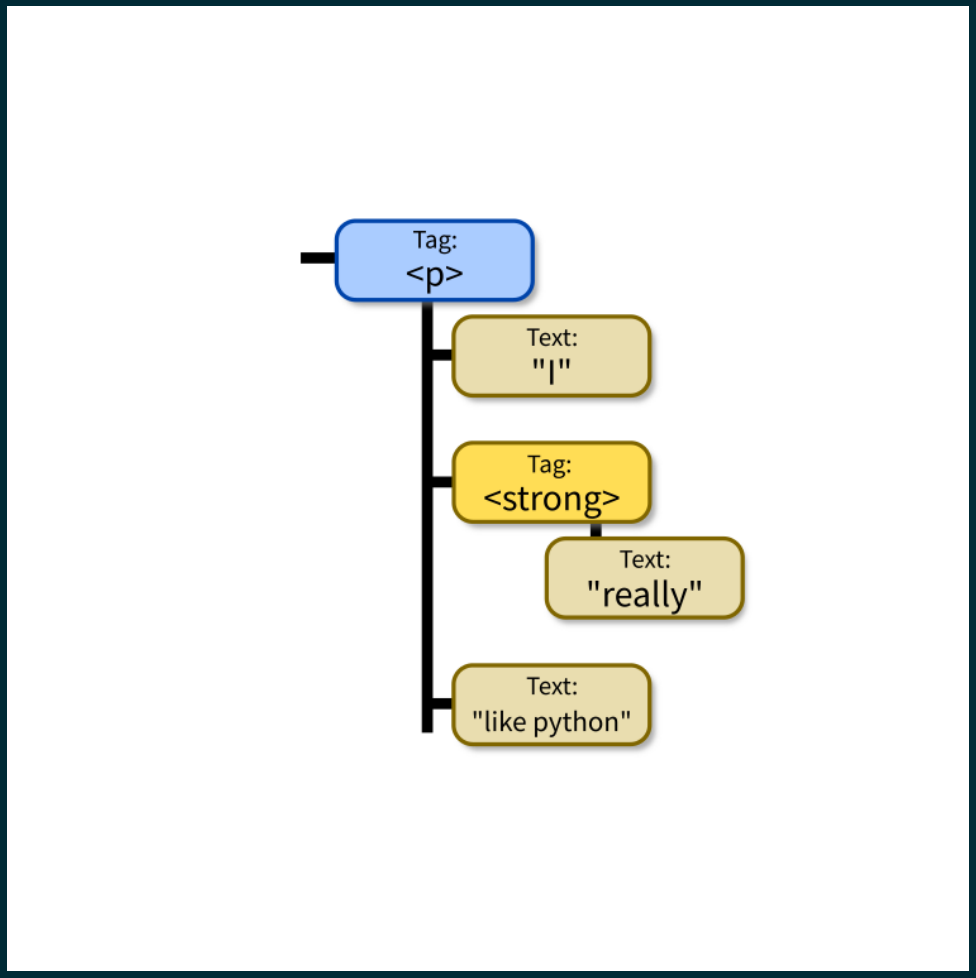
The formal specification has rules for for naming, accessing, and modifying parts of a document. JavaScript fully implements this specification.

```
<html><head><title>My title</title></head><body><h1>A heading</h1>
<a href="https://example.com">Link text</a></body></html>
```



Adapted from DOM illustration by Birger Eriksson (CC-BY-SA).

```
<p>I <strong>really</strong>like Python.</p>
```



Adapted from DOM illustration by Birger Eriksson (CC-BY-SA).

BEAUTIFUL SOUP

This package provides a module called `bs4` for turning HTML into a DOM-like data structure.

Widely used, e.g. at one point Reddit's backend software used it to select a representative image from a web page when a URL appeared in a post*.

Requires an HTML parser. We'll use `html.parser` from the standard library, but beautiful soup supports others.

* As of 2014. Perhaps they still use it?

MINIMAL SOUP

Parse HTML file into DOM:

```
from bs4 import BeautifulSoup

with open("lecture37.html") as fobj:
    soup = BeautifulSoup(fobj,"html.parser")
```

MINIMAL SOUP

Parse web page into DOM:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

with urlopen("https://example.com/") as response:
    soup = BeautifulSoup(response,"html.parser")
```

Be careful about the ethics of connecting to web servers from programs.

SCRAPING AND SPIDERS

A program that extracts data from HTML is a **scraper**

A program that visits all pages on a site is a **spider.**

All forms of automated access should:

- Allow the site to prioritize human users.
- Limit frequency of requests.
- Respect a site's Terms of Service (TOS).
- Respect the site's robots.txt automated access exclusion file, if they have one.

MINIMAL SOUP

Parse string into DOM:

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(
    "<p>The coffee was <strong>strong</strong>.</p>",
    "html.parser"
)
```

BS4 BASICS

```
str(soup) # show as HTML
soup.prettify() # prettier HTML
soup.title # first (and only) title tag
soup.p  # first p tag
soup.find("p") # first p tag (alternative)
soup.p.strong  # first strong tag within the first p tag
soup.find_all("a") # list of all a tags
```

WORKING WITH TAGS

```
str(tag) # HTML for this tag and everything inside it
tag.name # name of the tag, e.g. "a" or "ul"
tag.attrs # dict of tag's attributes
tag["href"] # get a single attribute
tag.text # All the text nodes inside tag, concatenated
tag.string # If tag has only text inside it, returns that text
           # But if it has other tags as well, returns None
tag.parent # enclosing tag
tag.contents # list of the children of this tag
tag.children # iterable of children of this tag
tag.banana # first descendant banana tag (sub actual tag name!)
tag.find(...) # first descendant meeting criteria
tag.find_all(...) # descendants meeting criteria
tag.find_next_sibling(...) # next sibling tag meeting criteria
```

SEARCHING

Arguments supported by all the `find*` methods:

```
tag.find_all(True) # all descendants
tag.find_all("tagname") # descendants by tag name
tag.find_all(href="https://example.com/") # by attribute
tag.find_all(class_="post") # by class
tag.find_all(re.compile("^fig")) # tag name regex match
tag.find_all("a",limit=15) # first 15 a tags
tag.find_all("a",recursive=False) # all a *children*
```

Also work with `find()`, `find_next_sibling()`, ...

SIMULATING CSS

`soup.select(SELECTOR)` returns a list of tags that match a CSS selector, e.g.

```
soup.select(".wide") # all tags of class "wide"

# ul tags within divs of class messagebox
soup.select("div.messagebox ul")
```

There are many CSS selectors and functions we haven't discussed, so this gives a powerful alternative search syntax.

```
# all third elements of unordered lists
soup.select("ul > li:nth-of-type(3)")
```

REFERENCES

- urllib documentation
- The Beautiful Soup documentation is beautifully clear.

REVISION HISTORY

- 2022-04-20 Last year's lecture on this topic finalized
- 2023-04-19 Updated for 2023