LECTURE 40

PARSING AND SCRAPING HTML

MCS 275 Spring 2021

Emily Dumas

LECTURE 40: PARSING AND SCRAPING HTML

Course bulletins:

- Project 4 is due 6pm CDT Friday April 30.
- Please install `beautifulsoup4` with

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

- Notebook: beautifulsoup-examples.ipynb

GETTING DATA FROM THE WEB

HTML is a language for making documents, meant to be displayed to humans. Avoid having programs read HTML if at all possible.

e.g. look for an API that serves the same data in a structured format (CSV, JSON, ...)

What do you do if there is no API, and you need to extract information from an HTML document?

Sigh with exasperation, then...

HTML PARSING

Level 0: Treat the HTML document as a string and use search operations (`str.find` or regexes) to locate something you care about, like `<title>`.

HTML is complicated, and this approach is very error-prone.

HTML PARSING

Level 1: Use a **parser** that knows how to recognize start/end tags, attributes, etc., and tell it what to do when it finds them (e.g. call this function...)

`html.parser` is in the standard library.

This approach is event-based. You specify functions to handle things when they are found, but you don't get an overall picture of the entire document.

HTML PARSING

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.

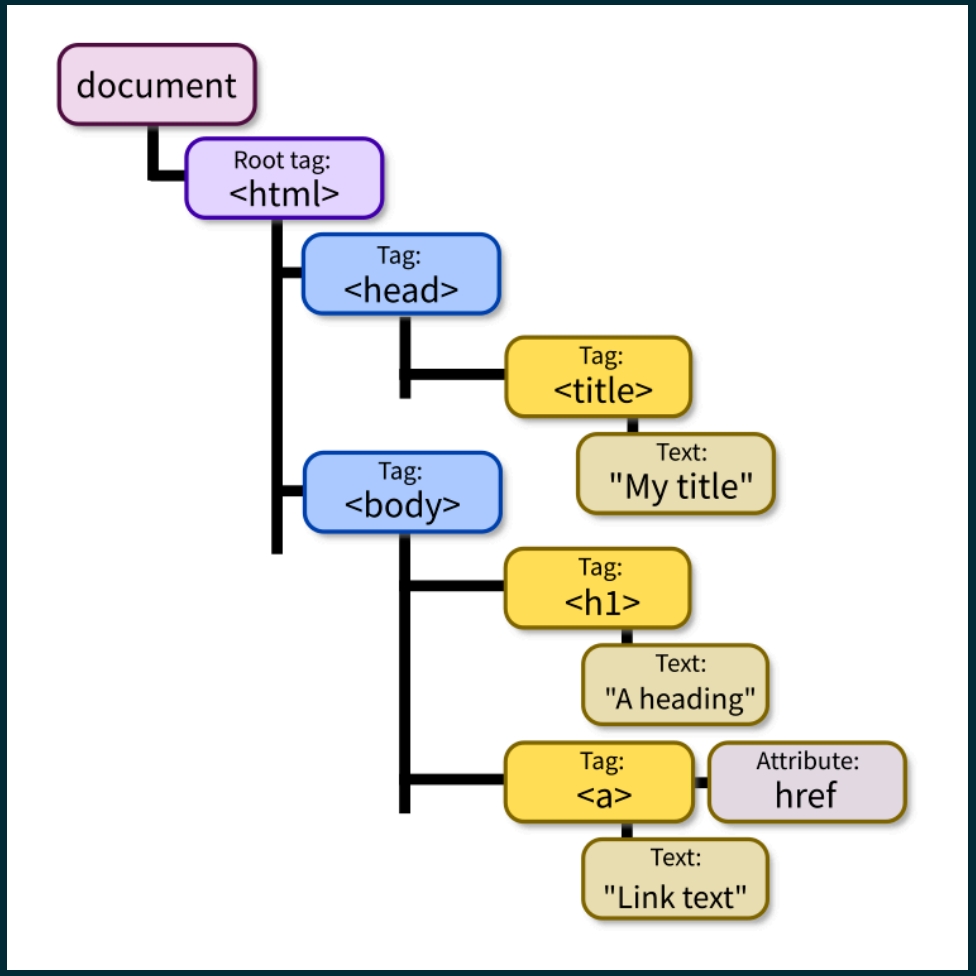
DOM

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.

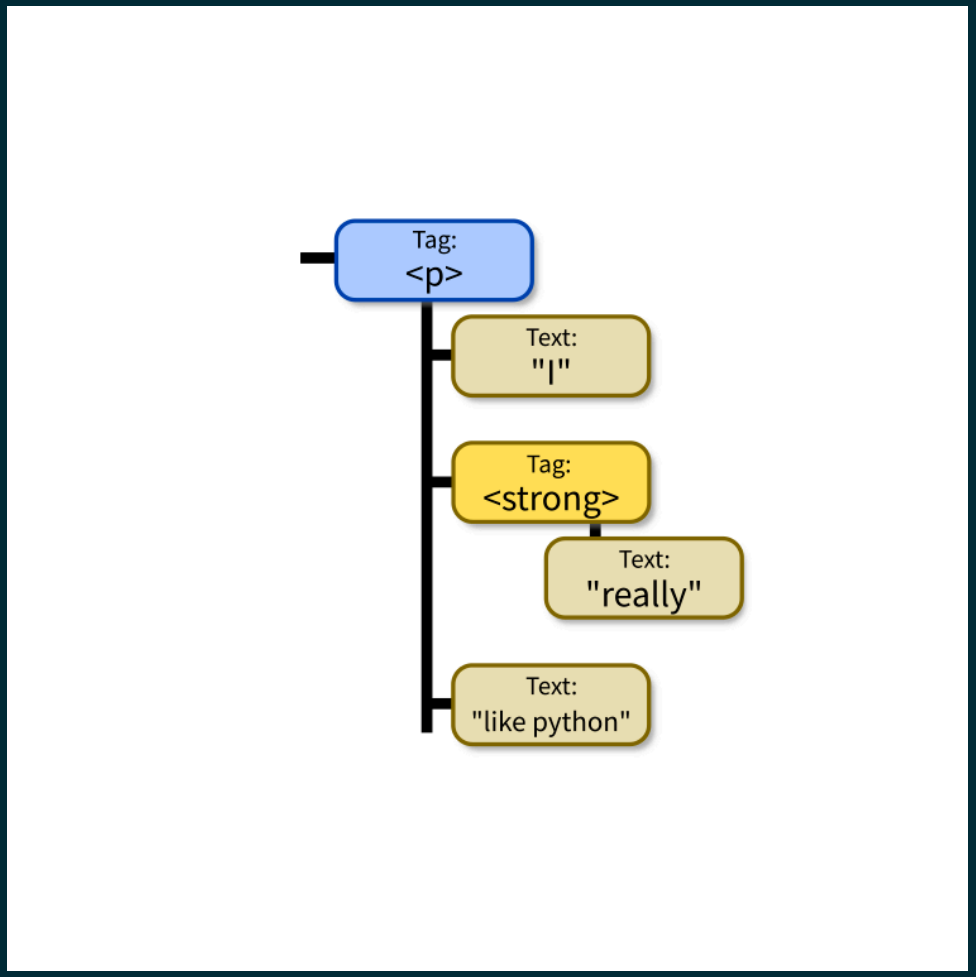
The formal specification has rules for for naming, accessing, and modifying parts of a document. JavaScript fully implements this specification.

```
<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. social network Reddit uses it* to select a representative image from a web page when a URL is submitted in a post.

Requires an HTML parser. We'll use `html.parser` from the standard library (slow but always available).

* At least that was the case in 2014; they may have switched to something else since then.

MINIMAL SOUP

Parse HTML file into DOM:

```
from bs4 import BeautifulSoup

with open("lecture40.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>That was <strong>durian</strong>?!</p>",
    "html.parser"
)
```

BS4 BASICS

```
str(soup) # the 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.em  # first em 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

- 2021-04-21 Typo fixes, notebook link
- 2021-04-21 Initial publication