DCS211: Lab 3 - Plan-of-Work Document
Teddy Shapiro & Liam Baron

Goal

Our goal will be to create student objects from the HTML documents and organize them by class year and by advisor. Each student object will need to include:

1) Name (“Last, First”)
2) Email
3) Class year
4) List of majors
5) List of minors
6) List of GECs
7) Advisor (“Last, First”)

Search within each HTML

- <table id="studentList"> contains the roster - we will parse through this table.
- Each row corresponds to one student
- Fields live in fixed columns
    - td[0] – includes a photo, will skip this one
    - td[1] – Name (“Last, First”)
    - td[3] – Class Year
    - td[5] – Email
    - td[6] – Majors (zero or more <abbr title="…">) - we will collect each title. Undeclared majors will appear <abbr title="0000"> and should become an empty string.
    - td[7] – minors (again, <abbr title="…"> list).
    - td[8] – GECs (again, <abbr title="…"> list).
    - td[9] – Advisors (“Last, First”)

Locating and extracting each piece

- To find the table → soup.find('table', id='studentList')
- To find the body rows → table.tbody.find_all('tr')
- Loop over rows → for each <tr>, call row.find_all('td') once; then map to the indexes listed above
- Grab each cell once for each row
    - Name: cells[1].get_text(strip=True)
    - Year: cells[3].get_text(strip=True)
    - Email: cells[5].find('a').get_text(strip=True)
    - Major, Minor, GEC: [abbr.get('title') for abbr in cells[“index”].find_all('abbr')]
    - Advisor: cells[9].find('span').get_text(strip=True)

- Build a Student from those pieces → Student: Student(name, email, year, majors, minors, gecs, advisor)

BeautifulSoup Functions

- To parse through the file, we'll use: BeautifulSoup(html_text, 'html.parser')
- To locate variables, we'll use find, find_all, get_text(strip=True), .get("title")
- Build two dicts:
    - by_year: dict[str, list[Student]] → append per student._year
    - by_advisor: dict[str, list[Student]] → append per student._advisor
- Handling edge cases:
    - We'll use empty strings for missing <a> in email cells
    - Empty minors/GEC cells will return empty lists
    - 000 major code will be treated as an empty major string

Interpreter Testing

Testing to ensure indexes are accurate and functions work properly:

```
>>> soup = BS(html, "html.parser")
>>> table = soup.find("table", id="studentList")
>>> len(table.tbody.find_all("tr"))
2
>>> r = table.tbody.find_all("tr")[0]
>>> cells = r.find_all("td")
>>> cells[1].get_text(strip=True)
'Beaniebag, Gumbs'
>>> cells[3].get_text(strip=True)
'2027'
>>> cells[5].find("a").get_text(strip=True)
'gbeanieb@bates.edu'
>>> [abbr.get("title") for abbr in cells[6].find_all("abbr")]
['Philosophy', 'Economics']
>>> [abbr.get("title") for abbr in cells[7].find_all("abbr")]
['Digital and Computational St.']
>>> [abbr.get("title") for abbr in cells[8].find_all("abbr")]
[]
>>> cells[9].find("span").get_text(strip=True)
'Shrout, Anelise'
```

Potential additional testing:

- Check that "studentList" is properly organized
    - Before implementing Beautiful Soup, create a fake student list in the Python code and print it to visualize the list formatting.
- Troubleshoot HTML extraction with Beautiful Soup
    - See if there are issues pulling items for "studentList"
    - First, use beautifulsoup to get the items for each student and print them to make sure they came out correctly before adding them to "studentList"

- Make a usage function that displays instructions for use of the program
    - Test that this is accurate and works correctly

Data Structure

- by_year: dict[str, list[Student]]
    - The first data structure will be by year, where the key is a four-digit year string
- by_advisor: dict[str, list[Student]]
    - The second will be by advisor, where the key is “First, Last”

Output

- If “Write CSV” is set to true by the user, the goal is to output a CSV per year, with columns containing [lname, fname, email, year, majors, minors, gecs, advisor]
    - Note: if the user specifies “Write CSV” = true, they will be prompted to enter a filename, which will be the HTML that the code reads to extract data, and write the yearly CSVs
- If “Write CSV” is set to false by the user, they will receive the same prompt, but after selecting a file, Python will instead display Pretty tables sorted by student, email, year, major(s), minor(s), and advisor, along with summary tables showing the count per year and count per advisor.

Summary

- In short, in parseMinors, we will find the table, loop over its rows—mapping cells and building Student objects —then append them into by_year and by_advisor, and finally return both dictionaries as a tuple.