# Module 2: Web Scraping

# JHU EP 605.256 – Modern Software Concepts in Python

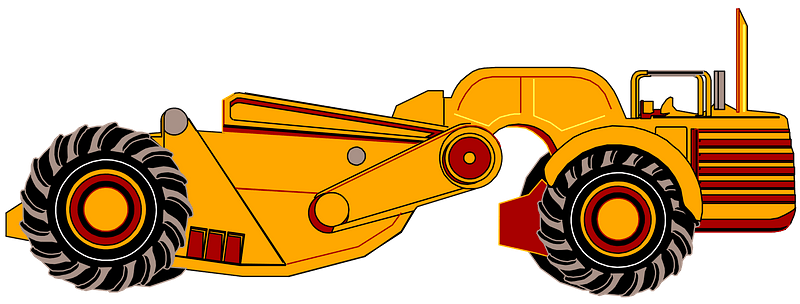
Introduction

This assignment is practices scraping data from websites. The site [grad cafe](https://www.thegradcafe.com/) allows people to upload information about programs they have been accepted / rejected / waitlisted from and basic applicant information, including date, start term, degree, program, and academic performance metrics. In this assignment, student developers (you!) are going to gather a copy of those freely uploaded student metrics and save it to be used in future requirements.

This will require you to manipulate urls, control HTML objects, and search within the structured data using several of the string methods taught in class this week. Your resultant output should clean dictionaries of information covering tens of thousands of student graduate school acceptance statistics – all ready to be analyzed in future assignments!

**Skills**: URL Management, HTML Searching Methods, Data Cleaning, JSON Data Object Storage





Assignment Overview

In this assignment you will:

1. Write a web scraper for [grad cafe](https://www.thegradcafe.com/) data about recent applicants and parse applicant data:
   1. Confirm the **robot.txt** file permits scraping.
   2. Use **urllib3** to request data from [grad cafe](https://www.thegradcafe.com/).
   3. Use **beautifulSoup/regex/string** search methods to find admissions data.
   4. Structure the data as a clean, formatted **json** data object.
2. Submit associated deliverables on-time and with the appropriate structure.

After this assignment, you will have a structured json object containing thousands of students uploaded data in an easy-to-use format. This data will be used for analysis on program statistics and admissions in later course modules.

1. Programming Assignment Requirements: Gathering the Data

A common way to organize requirements is into a **SHALL**, **SHOULD, SHALL NOT** list. Requirements for your homework assignment are under this structure and give you development freedom over how to implement – if it conforms to the Shall/Should/Shall Not list.

**SHALL:** A high priority requirement that must be implemented as an essential part of this requirement and will otherwise result in an unsatisfactory “program correctness” grade within our assignment matrix within the syllabus.

**SHOULD:** A low priority requirement that will result in a good/excellent “program correctness” grade if implemented.

**SHALL NOT:** A forbidden component of this assignment. This list does not include items specifically mentioned on the syllabus under academic integrity, but the expectation is that those areas are similarly respected.

For this assignment your solution:

* **SHALL** programmaticallypull data from [grad cafe](https://www.thegradcafe.com/) using python.
* **SHALL** only rely upon libraries explicitly covered within module 2 lecture.
* The data categories pulled **SHALL** include:
  + Program Name
  + University
  + Comments (if available)
  + Date of Information Added to Grad Café
  + URL link to applicant entry
  + Applicant Status
    - If Accepted: Acceptance Date
    - If Rejected: Rejection Date
  + Semester and Year of Program Start (if available)
  + International / American Student (if available)
  + GRE Score (if available)
  + GRE V Score (if available)
  + Masters or PhD (if available)
  + GPA (if available)
  + GRE AW (if available)
* **SHALL** use **urllib3** to carry out url management.
* **SHALL** use **json** to store data under file **applicant\_data.json** with reasonable object keys.
* **SHALL** include **at least 30,000 grad applicant entries**.
* **SHALL** include a README.
* **SHALL** be available on Github within a private repository called **jhu\_software\_concepts** within a folder named **module\_2.**
* **SHALL** comply with robots.txt (and include screenshot.jpg + README evidence that robots.txt was checked).
* **SHALL** include a requirements.txt file that allows complete reconstruction of your environment.
* **SHALL** use python 3.10+
* **SHOULD** use beautifulSoup / string methods / regex to find necessary data components
* **SHOULD** be written using either functions or class methods:
  + **scrape\_data()**: pulls data from grad cafe
  + **clean\_data()**: converts data into structured format
  + **save\_data()**: saves cleaned data into json file
  + **load\_data()**:loads cleaned data from json file
  + Other private methods can be used and should be indicated using **\_<private>()** with an underscore in front of the private method.
* **SHOULD** carry out scraping under file **scrape.py** and carry out data cleaning under file **clean.py.**
* **SHOULD** ensure data does not include any remnant HTML.
* **SHOULD** ensure unavailable data is maintained in a consistent format i.e. **None** or **“”**
* **SHOULD** handle removal of unexpected/ messy information.
* **SHOULD** be accurate information that is true to the website.
* **SHOULD** be well commented, clear, with appropriately named variables.
* **SHOULD NOT** use find/search methods that cannot be found within beautifulSoup / string methods / regex.

1. Programming Assignment Requirements: Cleaning the Data

When you started pulling the data, you probably noticed that the “program” field often mixes program and university names, and both show up in many different forms. The same institution might appear as JHU, Johns Hopkins, Johns Hopkins University, or even the misspelling John Hopkins. Messy labels like these make grouping, counting, joining, and any downstream analysis unreliable—especially when we’re dealing with more than 30,000 rows.

Cleaning this by hand isn’t realistic at our scale, and a rigid, rules-only script would be brittle. There are simply too many legitimate variants, abbreviations, and typos to anticipate. We also aren’t planning to spend money on a third-party API for this assignment, so sending each row to a hosted LLM service is off the table for now.

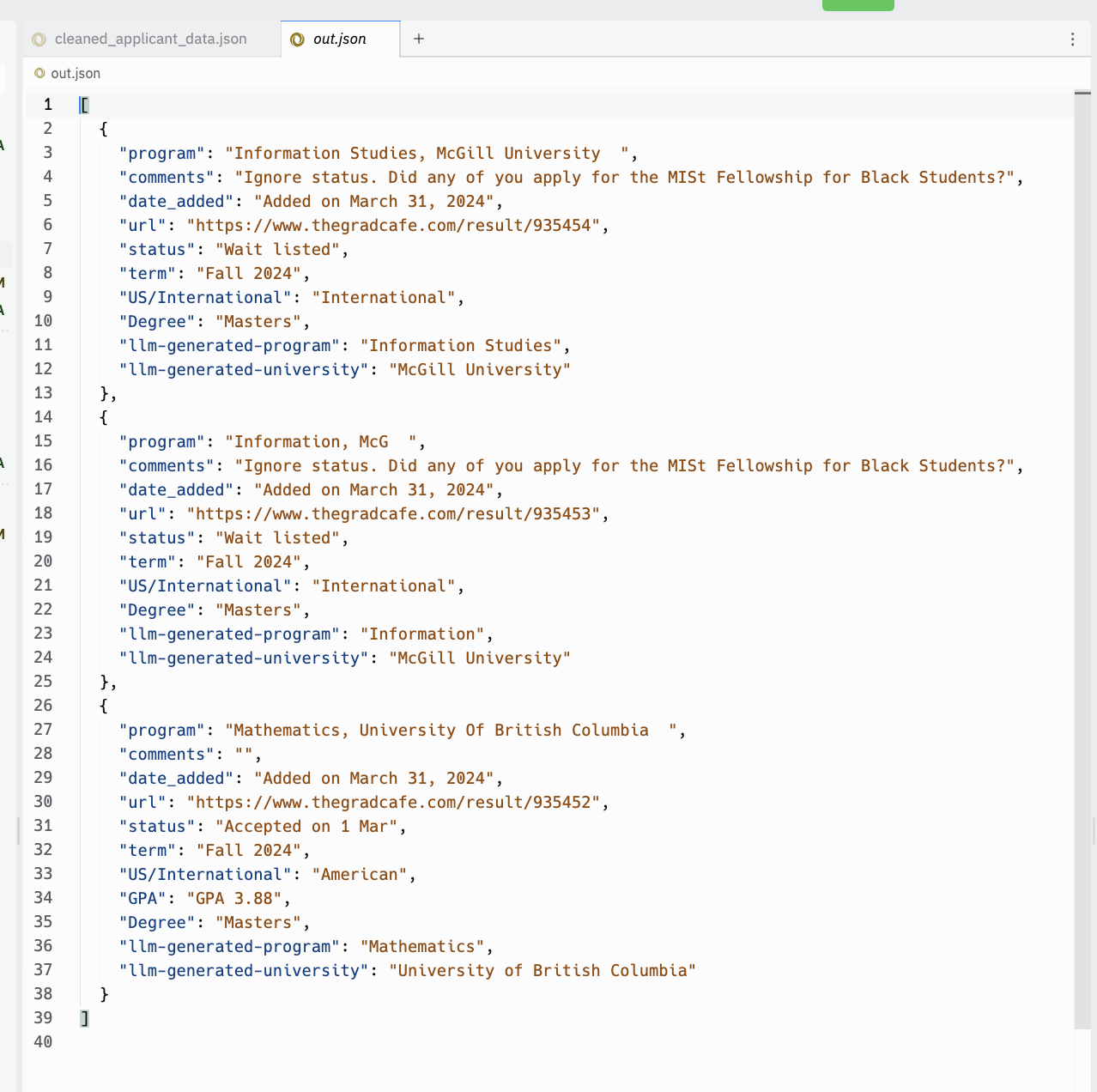
Our approach is to self-host a tiny local language model to do a first-pass standardization and then make that pass robust with a few pragmatic safeguards. The model proposes standardized values for both the program and the university. We then run those values through a post-processor that applies abbreviation and spelling fixes, maps near-matches to a canonical list of names, and uses lightweight fuzzy matching to catch small variations. This combination gives us fast, low-cost normalization that is “good enough” to analyze now and easy to improve later.

The goal isn’t perfection on the first attempt. We’ll generate standardized outputs, scan for outliers or obvious mistakes, and tighten things up by expanding the canonical lists or adding a couple of targeted fixes. Because the whole pipeline is local and repeatable, you can iterate quickly: update the canon files, rerun the tool, and watch the results converge.

To make this simple, I’ve bundled the code and a starter set of canonical lists in the zip files attached to the assignment. Download them, follow the README to set up the small local model, and run the standardizer on your dataset. You’ll end up with two additional columns—one for the cleaned program name and one for the cleaned university name—so the data is consistent enough for reliable analysis. When you’re done, submit your cleaned output along with any changes you made to the canonical lists and a brief note describing any systematic edge cases you found (update the README).

1. Add the zipped files to your repository area as a sub-package, i.e. module\_2/llm\_hosting
2. Enter the subfolder
3. pip install -r requirements.txt
4. Run python app.py --file “your\_part\_1.json” > out.json

You should then see the following:



As you can see, it is still not “perfect”, but we’re certainly making strides forward. Please ensure you maintain the original “program” name for reproducibility/traceability. Next week we’ll dig deeper into this data / see if we can find outliers / programs that are still incorrect after our efforts.

1. Deliverables

readme.txt

So-called “read me” files are a common way for developers to leave high-level notes about their applications. Here’s an example of a [README file](https://github.com/apache/spark/blob/master/README.md) for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name**: Your name and JHED ID
2. **Module Info**: The Module name/number along with the title of the assignment and its due date
3. **Approach**: a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn’t fully work.
4. **Known Bugs**: describe the areas, if any, where your code has any known bugs. If you’re asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.

Please post to your github ahead of deadline. Zipped github **module\_2** folder should also be submitted through Canvas so graders can use the timestamp of the submitted files. Instructors will check date / time of final push to confirm all materials were submitted on time.

Recap:

1. The SSH URL to your GitHub repository
2. scrape.py under **module\_2**
3. clean.py under **module\_2**
4. **llm\_hosting** folder with all the files I provided.
5. applicant\_data.json under **module\_2**
6. llm\_extend\_applicant\_data.json (our cleaned version) under **module\_2**
7. robots.txt screenshot under **module\_2**
8. README under **module\_2**

**Please let us know if you have any questions via Teams or email!**

# Sample Cleaned Data Output (Part I)

