Introduction

Welcome to Module 9! This week, we will learn about data retrieval from web sites. Web data extraction, also referred to as web scraping, is an automated process where data retrieval is performed based on criteria. Criteria represent the content of interest for your data retrieval. One important tool for web data extraction is Selenium. Selenium can be used from any Python application.

Topics and Learning Objectives

Topics

- Web Data Extraction, Web Scraping
- Web Scraping tools, Selenium
- Legal issues with regards to web scraping
- Countermeasures to prevent web scraping
- Automated web scraping using Python and Selenium

Learning Objectives

By the end of the module, you should be able to:

1. Get data from an API using the Requests package.
2. Describe the difference between consuming a first-party API and a third-party API.
3. Analyze web harvesting or web data extraction and open data consumption from different sources (Google, Reddit, etc.).

Readings & Resources

Reading

Required

- Ronquillo, A. (2022). Python's Requests library (Guide). Real Python.
- Video: Python Requests tutorial – How to call a weather API [11:28].
- Reitz, K., & Real Python. (2022). The Hitchhiker's guide to Python: HTML Scraping.

Web Scraping Intro

Web Data Extraction

- Web Scraping, web harvesting or web data extraction is data scraping used for extracting data from websites.
- Web scraping software directly accesses the WWW using HTTP or a web browser.
- Web scraping can be performed manually; it usually refers to an automated process that is implemented using a bot or web crawler.
- Scraping a web page involves fetching the web page and extracting information from the web page. Web crawlers are used for fetching web pages. Another program or script is used to extract targeted information from fetched web pages. An example of targeted information may be names, phone numbers and email addresses.

Web Scraping

- Web scraping is the process of automatically mining data or collecting information from the WWW.
- Web scraping is used as application components used for web indexing, web mining or data mining, online price change monitoring, weather data monitoring, real estate listings, website change detection and tracking online presences.
- Web pages are designed to be human readable and are not ideal to be accessed by automated tools such as bots. In order to facilitate machine readable web sites, specialized tools have been developed to aid with web scraping.
- There are two methods some websites can use to prevent web scraping.
    a. Detecting bots when fetching web pages
    b. Disallowing bots from crawling web pages
- Some sophisticated web scraping systems rely on using techniques in DOM parsing, computer vision and NLP to simulate human browsing to enable gathering web page content for offline parsing.

Legal Issues

- Legality of web scraping varies across the world.
- Web scraping may be against terms of use of web sites, but enforceability of these terms is unclear.
- Web site owners can use three major legal claims to prevent undesired web scraping:
    - Copyright infringement
    - Violation of the Computer Fraud and Abuse Act (United States)
    - Trespass to chattel

Methods to Prevent Web Scraping

- Blocking an IP address based on criteria such as geolocation and DNSRBL.
- Disable any web service API that the website's web scraper might use.

- Bots sometimes declare who they are and can be blocked on that basis.
- Bots can be blocked by monitoring excess traffic.
- Bots can be blocked with services like CAPTCHA.
- Locate bots with a honeypot in order to identify their IP address.
- Websites can declare if crawling is allowed or not in the “robots.txt” file.
- Load data straight into HTML DOM via AJAX and do not use DOM methods to display it. No visible data in the source means that it can’t be scraped.

Web Scraping Techniques and Tools

How does a Web Scraper Function?

There 3 steps are a high level overview of web scraping.

1. Making an HTTP request to a server.
2. Extract and parse the website’s code.
3. Save relevant data locally

Details of Scraping the Web

The steps below can be implemented in Python to scrape a web site.

1. Find URLs you want to scrape
2. Inspect the web pages
3. Identify the data you want to extract
4. Write the necessary code
5. Execute the code
6. Store the data generated from executing the code.

Web Scraping Techniques

Click on each of the techniques below to reveal more about them:

Click-n-reveal: **Human copy-and-paste**

The simplest form of web scraping is manually copying and pasting data from a web page into a text file or spreadsheet.

Click-n-reveal: **Text pattern matching**

A simple yet powerful approach to extracting information from web pages. It can be based on Unix commands (grep and regular expression scripts written in Perl or Python).

Click-n-reveal: **HTTP programming**

Static and dynamic web pages can be retrieved by posting HTTP request to web servers using socket programming.

Click-n-reveal: **HTML parsing**

Web sites have many web pages generated dynamically from a datastore. In such a case templates are used to generate web pages. In data mining, a program that detects templates is called a wrapper. Wrapper generation algorithms conform to a common template that can be easily identified in terms of a URL common scheme.

Click-n-reveal: **DOM parsing**

Using a full-fledged web browser, programs can retrieve dynamic content generated by client-side scripts. These browsers control and parse web pages into a DOM tree. Languages such as Xpath can be used to parse the resulting DOM tree.

Click-n-reveal: **Vertical aggregation**

These are tools or services created by third parties. These platforms create and moitor a multitude of “bots” for specific verticals with no “man in the loop” and no work related to a specific target site. The platform robustness is measured by the quality of information it retrieves.

Click-n-reveal: **Semantic annotation recognizing**

The pages being scraped my contain metadata or semantic markups and annotations. These can be used to locate specific data.

Click-n-reveal: **Computer vision web-page analysis**

The use of machine learning and computer vision used to identify and extract information from web pages by interpreting pages visually.

Web Scraping Tools

Click on each of the tools below to reveal more about them:

Click-n-reveal: **Scrapy**

A Python library that provides you with all the tools you need to quickly extract data from websites, analyze it and save it in a structure format (JSON, XML, CSV).

Click-n-reveal: **Selenium**

Selenium was originally created for automated testing of web applications. Selenium is used to fill out forms and navigate through hyperlinks. Selenium can execute JavaScript to scrape all populated (using AJAX calls) web pages.

Click-n-reveal: **Requests**

This is the most basic HTTP Python library. Requests allow users to send HTTP request and get responses in the form of HTML or JSON. This tools is simple to use.

Click-n-reveal: **Beautiful Soup**

This tool generates a parse tree for HTML and XML texts. This tools works well on web pages that are not well-structured.

Click-n-reveal: **LXML**

This Python library processes HTML data obtained from online sites. LXML is appropriate for scraping great amounts of data from any desired Internet database.

This Python library processes HTML data obtained from online sites. LXML is appropriate for scraping treat amounts of data from any desired internet database.

Using Selenium for Web Scraping

- Selenium can be installed using pip

```
pip install selenium
```

- Selenium requires a driver to imitate the actions of a real user as close as possible. You will need to set up a browser-specific driver for interfacing with Selenium.
  - For the Google Chrome web browser, you can download a ChromeDriver.
  - For the Mozilla Firefox web browser, you can download a GeckoDriver.
- Writing Python code starts with importing required modules.

```
from selenium import webdriver
# module for emulating keyboard actions
from selenium.webdriver.common.keys import Keys
# module for searching items using parameters
from selenium.webdriver.common.by import By
# module that waits for a web page to load
from selenium.webdriver.support.ui import WebDriverWait
# import module that issues instructions to wait for expected
conditions before executing scraping logic
from selenium.webdriver.support import expected_conditions as EC
```

- Selenium provides the WebDriver API, which defines the interface for imitating a real user's actions on a web browser.
- Every browser has its own unique implementation of the WebDriver called a driver.
- The code below creates an instance of the Chrome WebDriver.

```
driver =
webdriver.Chrome(service=Service(ChromeDriverManager().install()),
options=options)
```

- Note the above driver is generic. It should work for a Linux or a Windows Machine.
- The method driver.get() navigates to the web page needed for scraping data.
- Scraping data from Reddit. See below for code.

```
driver =
webdriver.Chrome(service=Service(ChromeDriverManager().install()),
options=options)
driver.get("https://www.reddit.com/")
```

- The WebDriver provides a wide range of find_element(s)_by_* methods to locate a single element of multiple elements on a web page. Please see Figure 9.1 below.
- You can use tag names, CSS selectors, Xpath, IDs, class names to select what will be extracted (see Figure 9.1).
- Figure 9.1 exhibits an image of Reddit rendered in a web browser. Note that the search box has a name attribute of q. This is seen in the inspector tab.

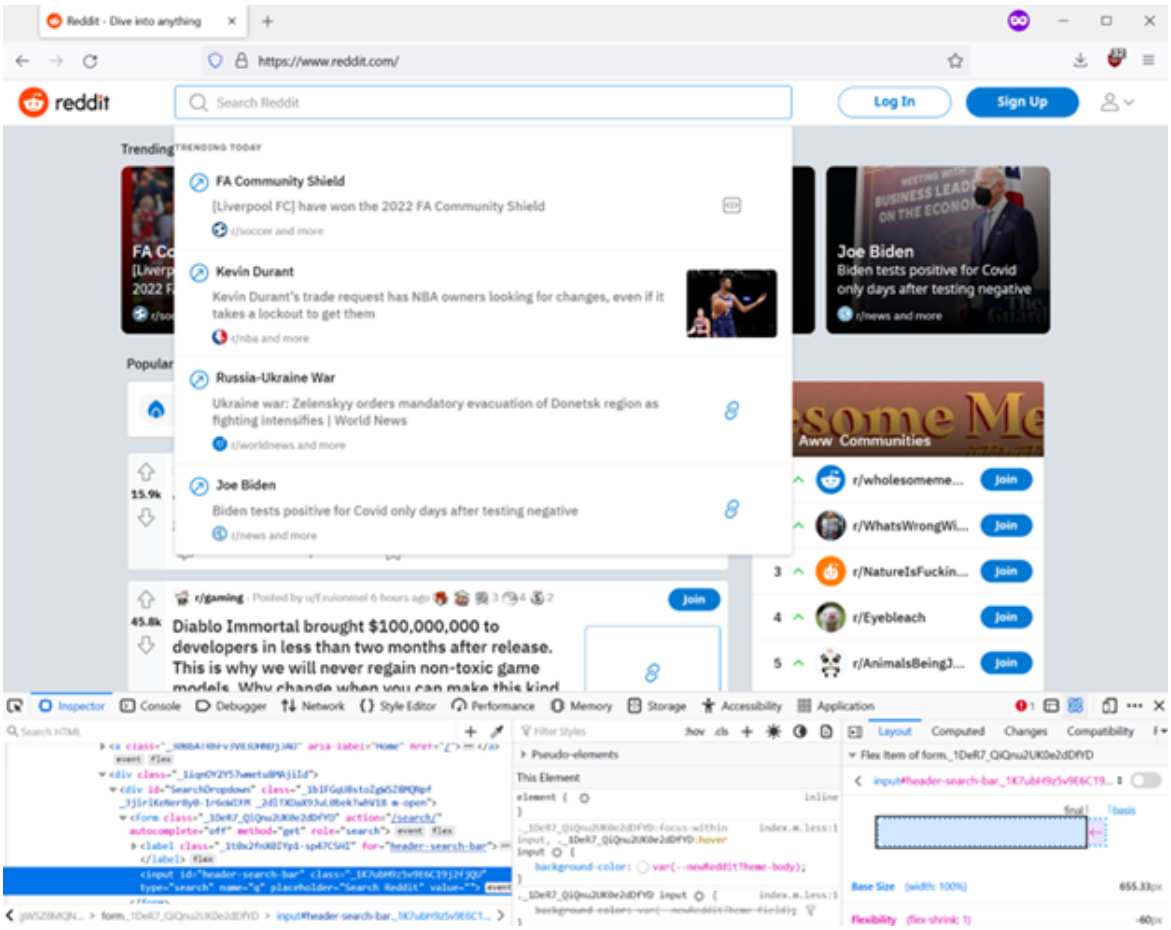

Figure 9.1. Reddit displayed in FireFox. Web developer tools are used to identify tags and their ids or names.

Source: Mihal Miu

- We can use find_element_by_name() to locate q. See below for code.

```
search = driver.find_element_by_name("q")
```

- The send_keys() method specifies the term we want to search in the input field.
- Keys.RETURN can be used to send the request.
- The code below is the same as entering the term “scraping” in the search box and pressing the enter key.

```
search.send_keys("scraping")
search.send_keys(Keys.RETURN)
```

Using Selenium for Web Scraping, cont.

- Modern websites use AJAX to load their content. This means when a browser load the page all HTML elements may not be present immediately to be visible by the user. Elements are loaded at different intervals making scraping difficult.
- Selenium WebDriver provides the wait feature to allow us to scrape web pages loaded with AJAX.
- We need to wait for the element we want to be present before proceeding with the code execution for extraction. This is accomplished using a combination of WebDriverWait() and ExpectedCondition() method calls.
- We want to examine the search results. We notice that all the posts are enclosed in a div. How do we identify this div? Please see Figure 9.2 below.
- The enclosing results are in a div and this div has an unique class. The class is “QBfRw7Rj8UkxybFpX-USO” (see Figure 9.2).

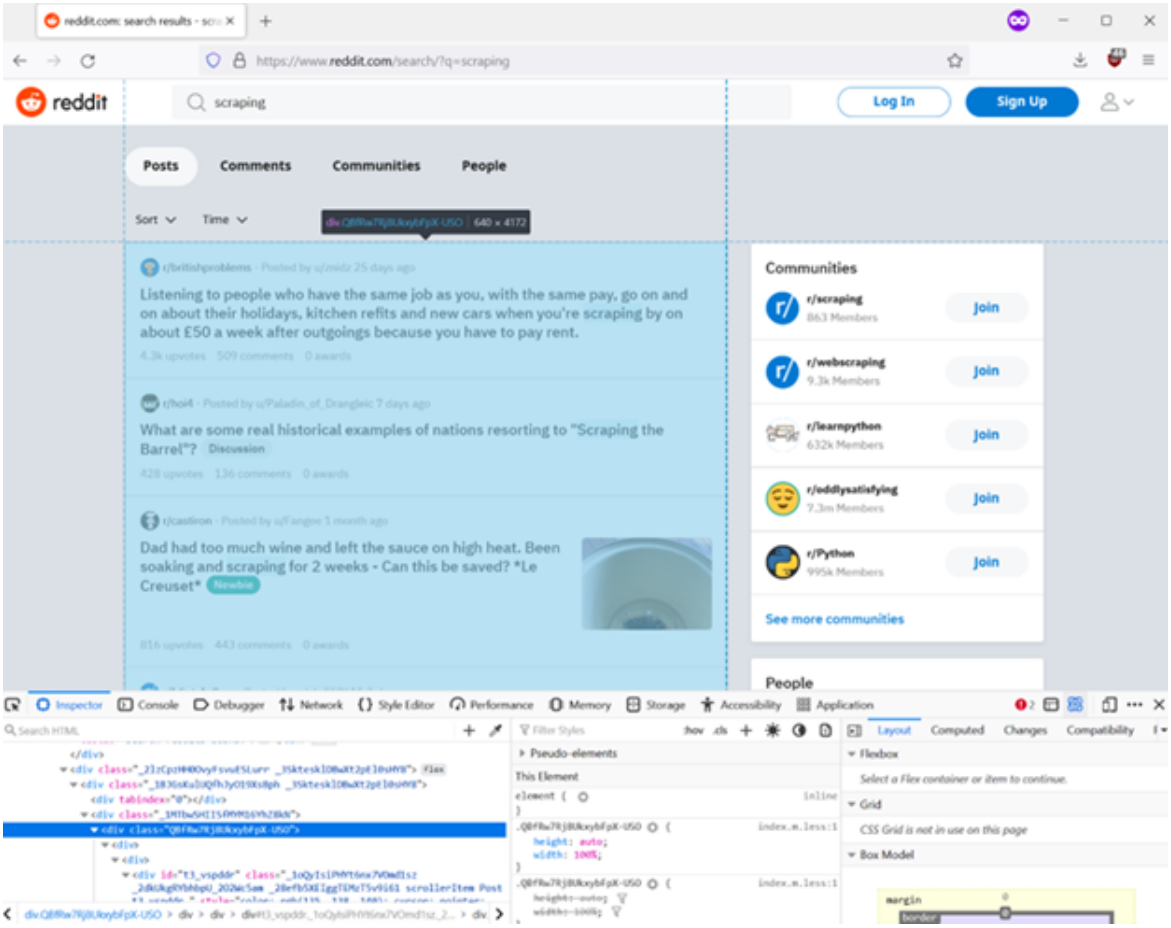

Figure 9.2. Reddit displayed in FireFox. Web developer tools are used to identify tags and their classes.

Source: Mihal Miu

- Below is the code to locate the search results.

```
search_results = WebDriverWait(driver, 20).until(
EC.presence_of_element_located((By.CLASS_NAME, " QBfRw7Rj8UkxybFpX-USO "))
)
```

- Now that we have search results, we can scrape information returned from the search results.
- Note that each result is wrapped in an h3 tag and a “eYtD2XCVieq6emjKBH3m” class. Furthermore, each heading is enclosed in a span tag. Please see Figure 9.3 below.

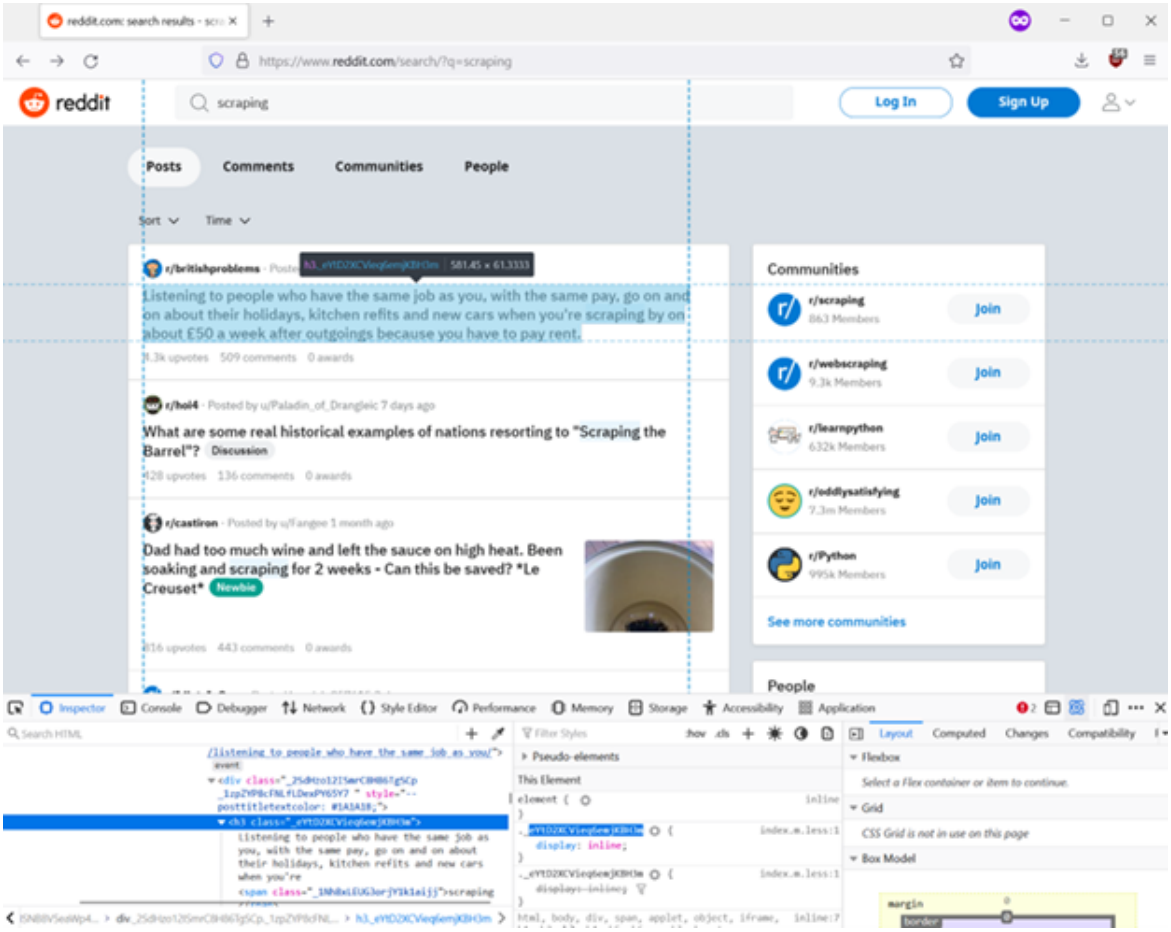

Figure 9.3. Reddit displayed in FireFox. Web developer tools are used to identify specific tags such as an h3 tag.

Source: Mihal Miu

- We can select all of the posts' headings and store them in a list.

```
posts =
search_results.find_elements_by_css_selector("h3._eYtD2XCVieq6emjKBH3m")
```

- We can iterate over each heading and output their content.

```
for post in posts:
  header = post.find_element_by_tag_name("span")
  print(header.text)
```

- Now that we have extracted data we need to exit the browser.

```
driver.quit()
```

Video

Please watch the following video, Web Harvesting Using Selenium. This video demonstrates how to scrape web pages using Selenium.

Web Harvesting Using Selenium
TMU Video

Summary

Web Data Retrieval should be an automated process. This implies that a Python application (using Selenium to aid with the web scraping) is created that targets topics of interest. This represents an automated process that has the ability to run at any time and retrieve data based on our topics of interest. Google and Reddit are web sites that provide current data as well as historical data. Running our web scraping application at different times in a year may reveal new facts or trends.

Assessments

Each week, you'll find here reminders of assignments or labs that you should be working on. For Week 9:

- Lab 9: Data harvesting from Google (5% of the final grade) is due in 7 days, i.e., the end of day Friday of Week 9.

Reminder

You have a weekly Zoom session. The date/timing for your Zoom session can be found in the Course Schedule in the Course Outline. The link to the Zoom session will be provided separately to you by your instructor. The Zoom session will consist of 20–30 minutes of lecture, followed by questions and answers (Q&A), followed by a lab period. You may ask questions about the previous or current week's content or labs.

Please bring any questions to the weekly Zoom session. However, if you have any questions during the week outside of the Zoom session, you may post them in the Discussion Board. Your instructor may respond in the Discussion board or during the weekly Zoom session.

Click-n-reveal: **Where is the Discussion board?**

Click on the "Communication" area at the top main course menu, and select Discussions from the dropdown menu.

References

None for this week.