Web Scraping Tutorial

Dasha Ageikina
Glynmoran
LinkedIn

We’ll cover

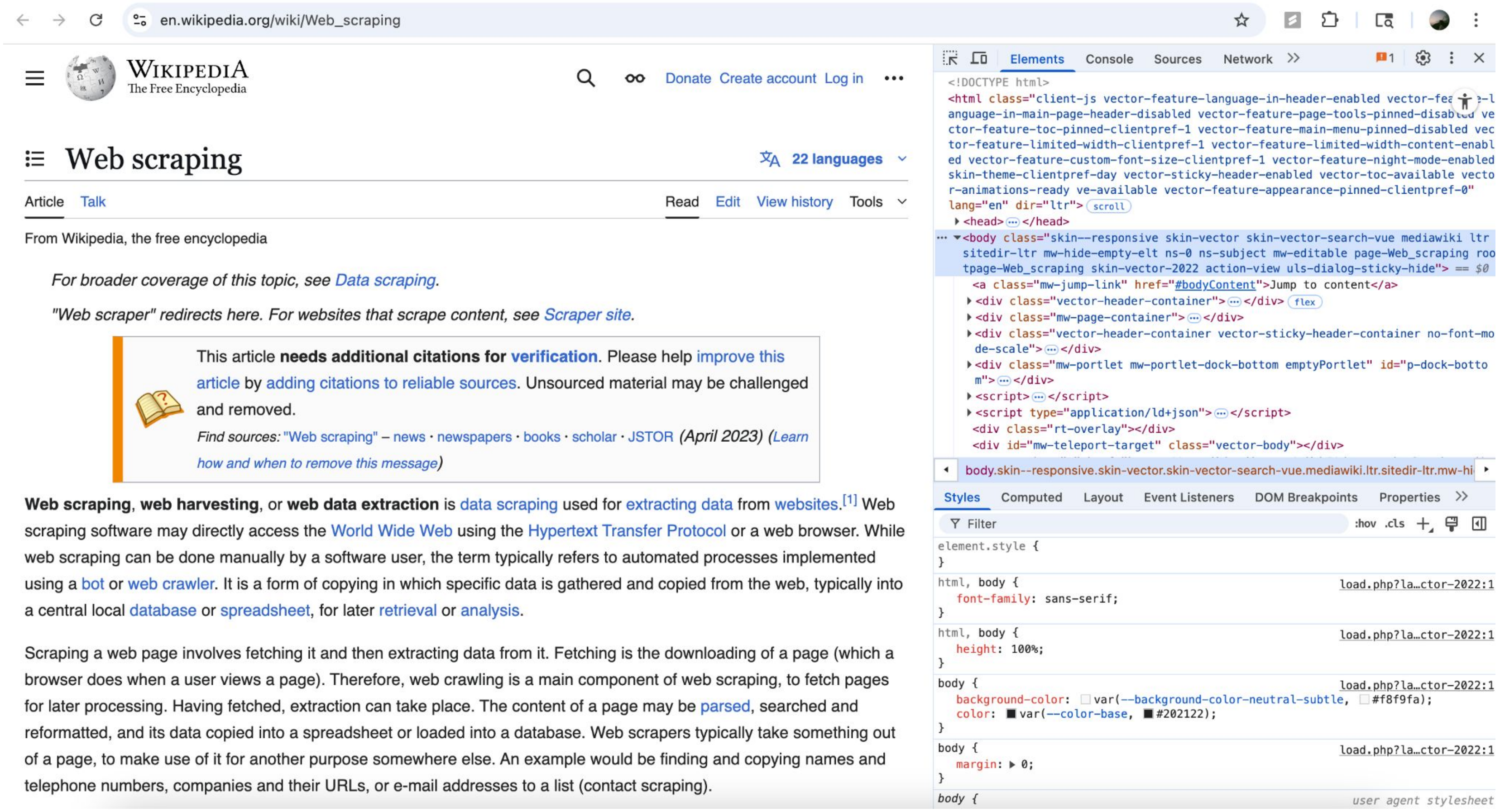
- The basics of web scraping
- 3 web scraping examples in **Python**
    - You can do web scraping in **R** too but I recommend learning **Python** because it is:
        - a universal language for most industry data jobs
        - easy to debug and troubleshoot and has many good resources online
        - more efficient than **R**
- Cookies and other considerations in web scraping
- Challenges

Disclaimer: This is not a comprehensive tutorial on web scraping since it’s a very big topic. I focus on what was useful for me. There may be better/more efficient solutions for some projects – please share if you find those!

Definitions

- **Web scraping** - automated process of extracting data from websites.
- **HTTP (HyperText Transfer Protocol)** – protocol used by the web to transfer data between clients (your browser or Python program) and servers (website hosts).
- **URL (Uniform Resource Locator)** - web address (link to the website).
- **HTML (HyperText Markup Language)** is a standard language used to create websites. To extract data from websites, you need to learn how to access website elements in **HTML.** HTML elements are delineated by **tags**.
- **CSS (Cascading Style Sheets)** is a language for formatting HTML elements (colors, fonts, etc.)

To see a website in HTML using Google Chrome on **Mac**: View → Developer → Developer Tools; **Windows**: three dots in top-right corner → More tools → Developer Tools



Web scraping

From Wikipedia, the free encyclopedia

For broader coverage of this topic, see Data scraping.

"Web scraper" redirects here. For websites that scrape content, see Scraper site.

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Web scraping" – news · newspapers · books · scholar · JSTOR *(April 2023) (Learn how and when to remove this message)*

Web scraping, **web harvesting**, or **web data extraction** is data scraping used for extracting data from websites.[1] Web scraping software may directly access the World Wide Web using the Hypertext Transfer Protocol or a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.

Scraping a web page involves fetching it and then extracting data from it. Fetching is the downloading of a page (which a browser does when a user views a page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Having fetched, extraction can take place. The content of a page may be parsed, searched and reformatted, and its data copied into a spreadsheet or loaded into a database. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. An example would be finding and copying names and telephone numbers, companies and their URLs, or e-mail addresses to a list (contact scraping).

```
<!DOCTYPE html>
<html class="client-js vector-feature-language-in-header-enabled vector-fea  e-l
anguage-in-main-page-header-disabled vector-feature-page-tools-pinned-disab   ve
ctor-feature-toc-pinned-clientpref-1 vector-feature-main-menu-pinned-disabled vec
tor-feature-limited-width-clientpref-1 vector-feature-limited-width-content-enabl
ed vector-feature-custom-font-size-clientpref-1 vector-feature-night-mode-enabled
skin-theme-clientpref-day vector-sticky-header-enabled vector-toc-available vecto
r-animations-ready ve-available vector-feature-appearance-pinned-clientpref-0"
lang="en" dir="ltr"> scroll
 ▸<head> … </head>
… ▾<body class="skin--responsive skin-vector skin-vector-search-vue mediawiki ltr
   sitedir-ltr mw-hide-empty-elt ns-0 ns-subject mw-editable page-Web_scraping roo
   tpage-Web_scraping skin-vector-2022 action-view uls-dialog-sticky-hide"> == $0
     <a class="mw-jump-link" href="#bodyContent">Jump to content</a>
    ▸<div class="vector-header-container"> … </div> flex
    ▸<div class="mw-page-container"> … </div>
    ▸<div class="vector-header-container vector-sticky-header-container no-font-mo
     de-scale"> … </div>
    ▸<div class="mw-portlet mw-portlet-dock-bottom emptyPortlet" id="p-dock-botto
     m"> … </div>
    ▸<script> … </script>
    ▸<script type="application/ld+json"> … </script>
     <div class="rt-overlay"></div>
     <div id="mw-teleport-target" class="vector-body"></div>
body.skin--responsive.skin-vector.skin-vector-search-vue.mediawiki.ltr.sitedir-ltr.mw-hi
```

```
element.style {
}
html, body {                                            load.php?la…ctor-2022:1
   font-family: sans-serif;
}
html, body {                                            load.php?la…ctor-2022:1
   height: 100%;
}
body {                                                  load.php?la…ctor-2022:1
   background-color: □ var(--background-color-neutral-subtle, □ #f8f9fa);
   color: ■ var(--color-base, ■ #202122);
}
body {                                                  load.php?la…ctor-2022:1
   margin: ▸ 0;
}
body {                                                  user agent stylesheet
```

Some HTML tags

<html>: The root element wrapping the entire HTML document.

<head>: Meta-information about the document, such as its title and links to scripts or stylesheets.

<body>: All the visible content of the web page, such as text, images, and links.

<nav>: Navigation menus or links.

<div>: Container used to group HTML elements together.

<table>: Tabular data.

<tr>: Row of a table.

<td>: Element in a row of a table.

<a>: Contains hyperlinks to other pages or resources

The main web scraping steps in Python

1. **Send an HTTP request** (often called get request) to the URL of the website and save the response to an object.
2. This object will contain the HTML content of the website - it will have a complicated non-string structure.
3. We can **parse this HTML object** using one of the libraries such as:
   a. lxml
   b. BeautifulSoup
   c. Html5lib
4. Sometimes, we need to interact with the website and use Selenium

Problem 1 - Download data from files with links

Sometimes we can download data directly.

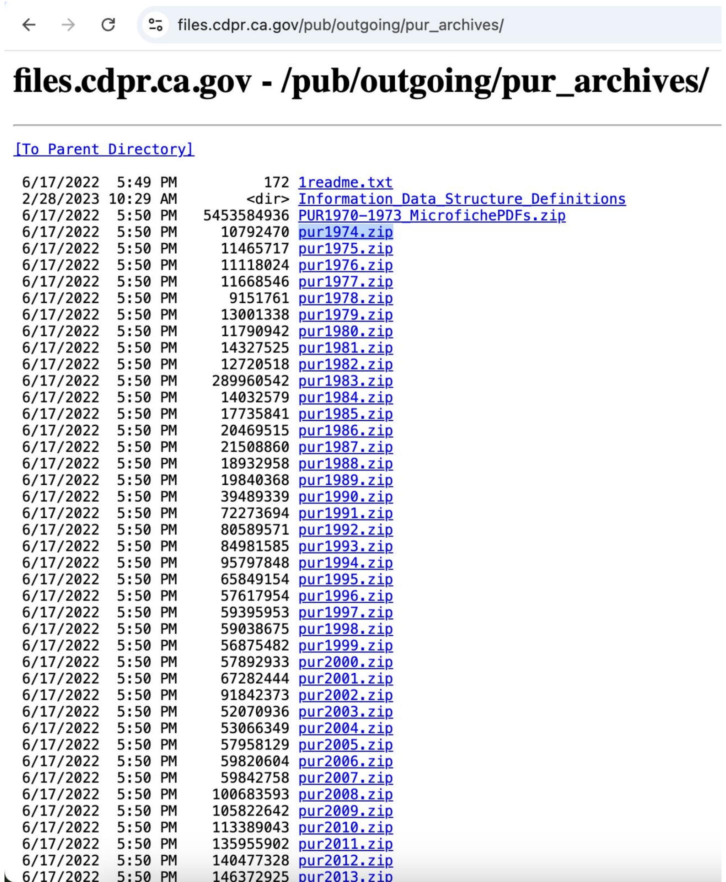
Here, we can simply use the links to the files.

For instance, a link to the file “pur1974.zip” will look like this:

https://files.cdpr.ca.gov/pub/outgoing/pur_archives/pur1974.zip

Similarly, a link to “pur1975.zip” will be:

https://files.cdpr.ca.gov/pub/outgoing/pur_archives/pur1975.zip

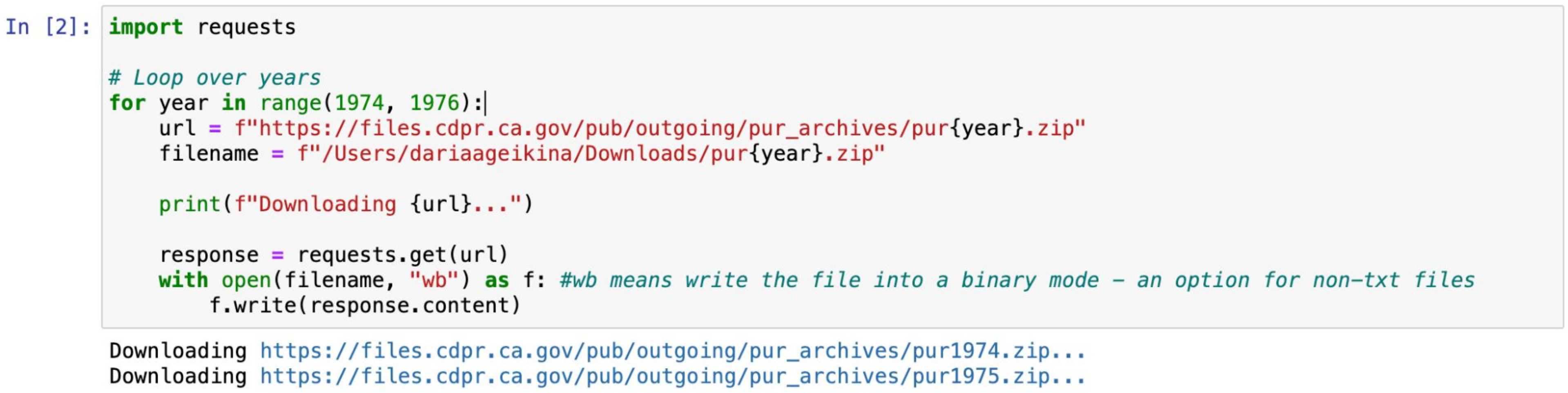


files.cdpr.ca.gov/pub/outgoing/pur_archives/

files.cdpr.ca.gov - /pub/outgoing/pur_archives/

```
[To Parent Directory]

 6/17/2022  5:49 PM          172 1readme.txt
 2/28/2023 10:29 AM        <dir> Information_Data_Structure_Definitions
 6/17/2022  5:50 PM   5453584936 PUR1970-1973_MicroficheePDFs.zip
 6/17/2022  5:50 PM     10792470 pur1974.zip
 6/17/2022  5:50 PM     11465717 pur1975.zip
 6/17/2022  5:50 PM     11118024 pur1976.zip
 6/17/2022  5:50 PM     11668546 pur1977.zip
 6/17/2022  5:50 PM      9151761 pur1978.zip
 6/17/2022  5:50 PM     13001338 pur1979.zip
 6/17/2022  5:50 PM     11790942 pur1980.zip
 6/17/2022  5:50 PM     14327525 pur1981.zip
 6/17/2022  5:50 PM     12720518 pur1982.zip
 6/17/2022  5:50 PM    289960542 pur1983.zip
 6/17/2022  5:50 PM     14032579 pur1984.zip
 6/17/2022  5:50 PM     17735841 pur1985.zip
 6/17/2022  5:50 PM     20469515 pur1986.zip
 6/17/2022  5:50 PM     21508860 pur1987.zip
 6/17/2022  5:50 PM     18932958 pur1988.zip
 6/17/2022  5:50 PM     19840368 pur1989.zip
 6/17/2022  5:50 PM     39489339 pur1990.zip
 6/17/2022  5:50 PM     72273694 pur1991.zip
 6/17/2022  5:50 PM     80589571 pur1992.zip
 6/17/2022  5:50 PM     84981585 pur1993.zip
 6/17/2022  5:50 PM     95797848 pur1994.zip
 6/17/2022  5:50 PM     65849154 pur1995.zip
 6/17/2022  5:50 PM     57617954 pur1996.zip
 6/17/2022  5:50 PM     59395953 pur1997.zip
 6/17/2022  5:50 PM     59038675 pur1998.zip
 6/17/2022  5:50 PM     56875482 pur1999.zip
 6/17/2022  5:50 PM     57892933 pur2000.zip
 6/17/2022  5:50 PM     67282444 pur2001.zip
 6/17/2022  5:50 PM     91842373 pur2002.zip
 6/17/2022  5:50 PM     52070936 pur2003.zip
 6/17/2022  5:50 PM     53066349 pur2004.zip
 6/17/2022  5:50 PM     57958129 pur2005.zip
 6/17/2022  5:50 PM     59820604 pur2006.zip
 6/17/2022  5:50 PM     59842758 pur2007.zip
 6/17/2022  5:50 PM    100683593 pur2008.zip
 6/17/2022  5:50 PM    105822642 pur2009.zip
 6/17/2022  5:50 PM    113389043 pur2010.zip
 6/17/2022  5:50 PM    135955902 pur2011.zip
 6/17/2022  5:50 PM    140477328 pur2012.zip
 6/17/2022  5:50 PM    146372925 pur2013.zip
```

Problem 1 - Python solution



```
In [2]: import requests

        # Loop over years
        for year in range(1974, 1976):
            url = f"https://files.cdpr.ca.gov/pub/outgoing/pur_archives/pur{year}.zip"
            filename = f"/Users/dariaageikina/Downloads/pur{year}.zip"

            print(f"Downloading {url}...")

            response = requests.get(url)
            with open(filename, "wb") as f: #wb means write the file into a binary mode - an option for non-txt files
                f.write(response.content)
```

```
Downloading https://files.cdpr.ca.gov/pub/outgoing/pur_archives/pur1974.zip...
Downloading https://files.cdpr.ca.gov/pub/outgoing/pur_archives/pur1975.zip...
```

Problem 2 - parse no-link elements from a website

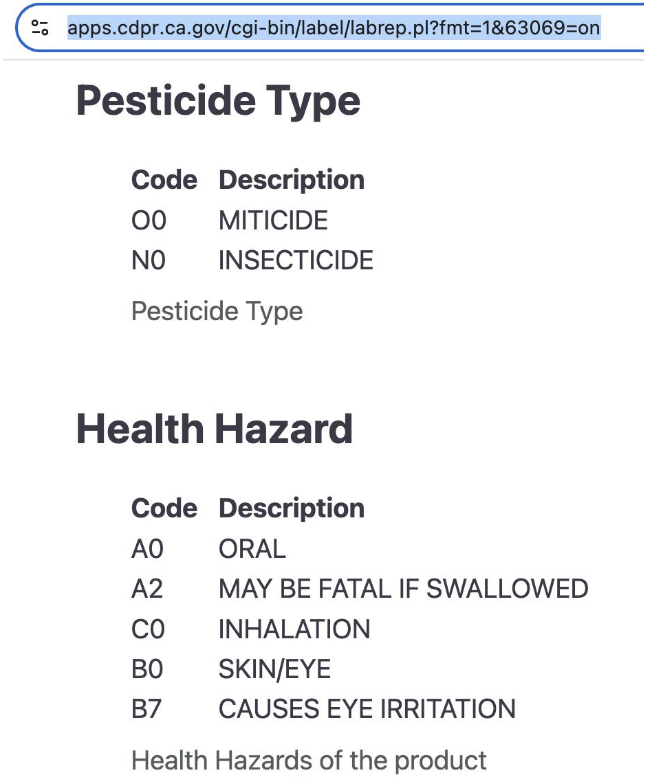
We want to collect the data on pesticides.

Consider a link to a report on a pesticide.

*We need to extract the data on its **type** - is it herbicide, insecticide, fungicide?*

Here, we need to use Google Chrome browser:

1. Open the link in Google Chrome
2. Right-click the element of interest → Inspect
3. Right-click the source-code of the element in the new panel on the right → Copy → …


apps.cdpr.ca.gov/cgi-bin/label/labrep.pl?fmt=1&63069=on

Pesticide Type

Code	Description
O0	MITICIDE
N0	INSECTICIDE

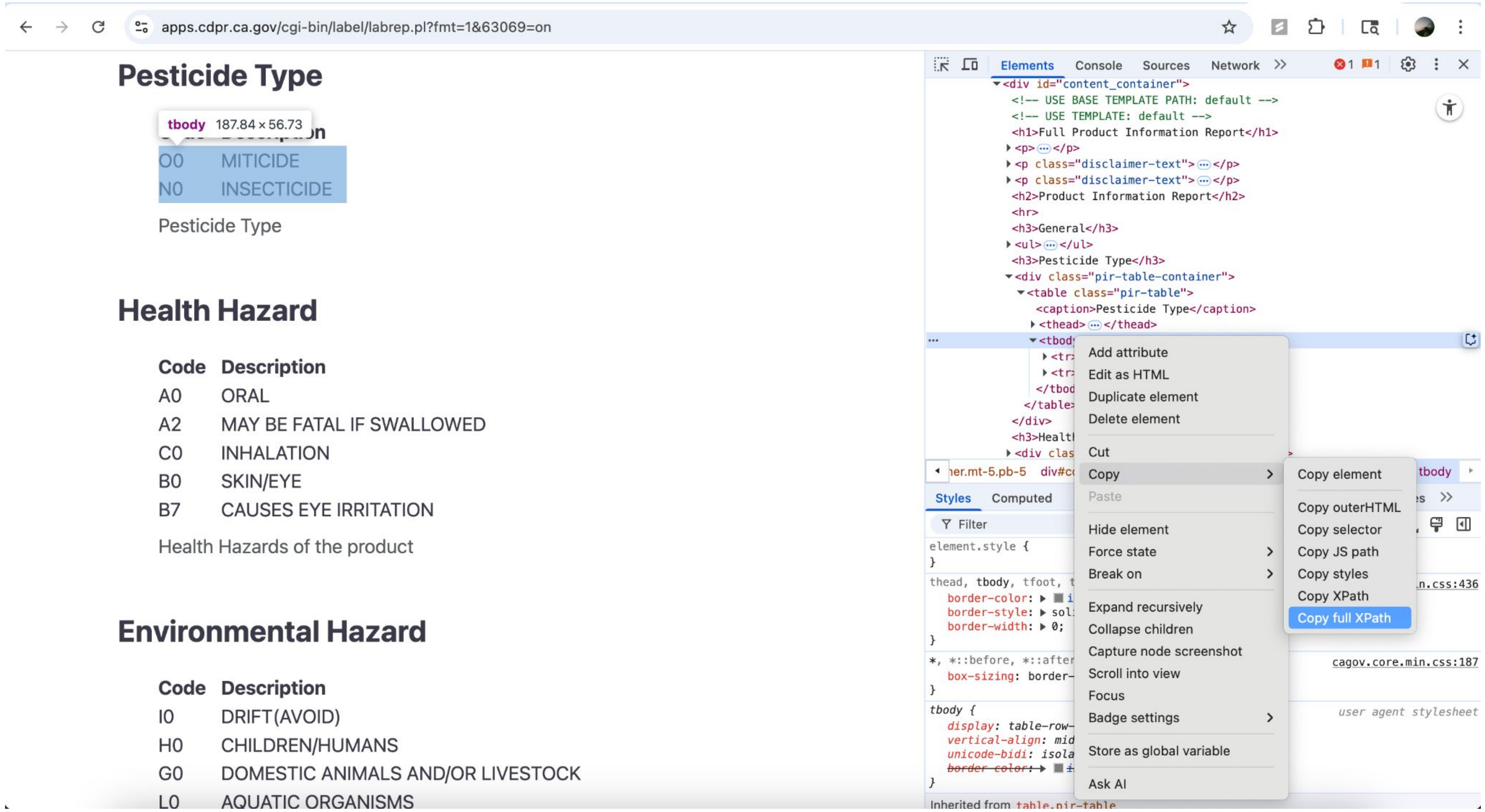
Pesticide Type

Health Hazard

Code	Description
A0	ORAL
A2	MAY BE FATAL IF SWALLOWED
C0	INHALATION
B0	SKIN/EYE
B7	CAUSES EYE IRRITATION

Health Hazards of the product

Problem 2 - parse no-link elements from a website



apps.cdpr.ca.gov/cgi-bin/label/labrep.pl?fmt=1&63069=on

Pesticide Type

tbody 187.84 × 56.73

Code	Description
O0	MITICIDE
N0	INSECTICIDE

Pesticide Type

Health Hazard

Code	Description
A0	ORAL
A2	MAY BE FATAL IF SWALLOWED
C0	INHALATION
B0	SKIN/EYE
B7	CAUSES EYE IRRITATION

Health Hazards of the product

Environmental Hazard

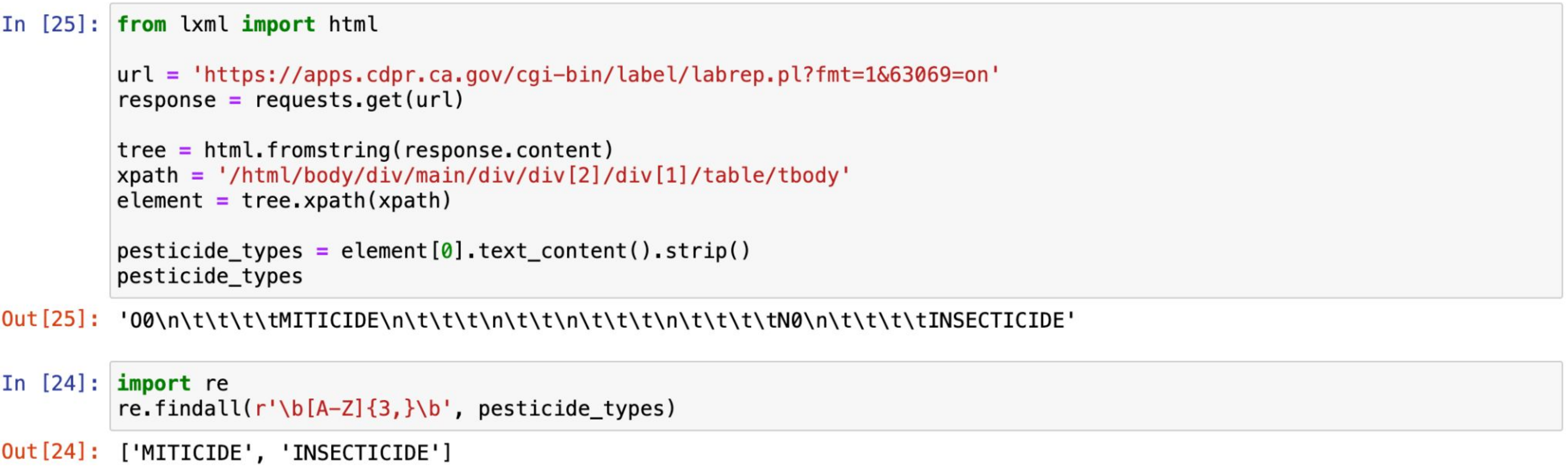
Code	Description
I0	DRIFT(AVOID)
H0	CHILDREN/HUMANS
G0	DOMESTIC ANIMALS AND/OR LIVESTOCK
L0	AQUATIC ORGANISMS

```
<div id="content_container">
  <!-- USE BASE TEMPLATE PATH: default -->
  <!-- USE TEMPLATE: default -->
  <h1>Full Product Information Report</h1>
  <p>…</p>
  <p class="disclaimer-text">…</p>
  <p class="disclaimer-text">…</p>
  <h2>Product Information Report</h2>
  <hr>
  <h3>General</h3>
  <ul>…</ul>
  <h3>Pesticide Type</h3>
  <div class="pir-table-container">
    <table class="pir-table">
      <caption>Pesticide Type</caption>
      <thead>…</thead>
      <tbody
        <tr
        <tr
      </tbod
    </table
  </div>
  <h3>Healt
  <div clas
```

```
element.style {
}
thead, tbody, tfoot, t
  border-color: ▸ i
  border-style: ▸ sol
  border-width: ▸ 0;
}
*, *::before, *::after                    cagov.core.min.css:187
  box-sizing: border-
}
tbody {                                   user agent stylesheet
  display: table-row-
  vertical-align: mid
  unicode-bidi: isola
  ~~border-color: ▸ i~~
}
Inherited from table.pir-table
```

Problem 2 - Python solution 1 - full XPath + lxml library

Option 1: Copy full XPath - kind of an address of the element - best option when you can extract your elements using one xpath



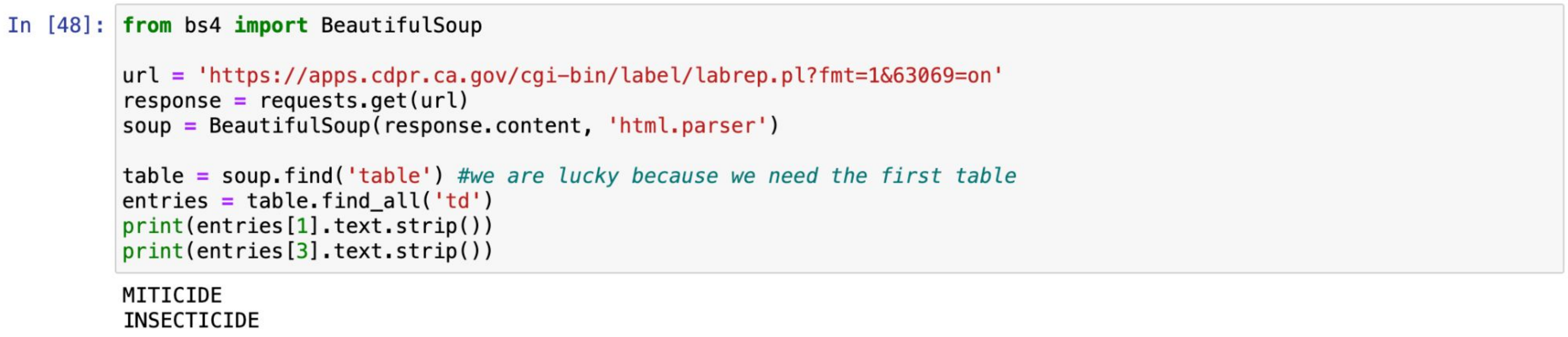
```
In [25]: from lxml import html

         url = 'https://apps.cdpr.ca.gov/cgi-bin/label/labrep.pl?fmt=1&63069=on'
         response = requests.get(url)

         tree = html.fromstring(response.content)
         xpath = '/html/body/div/main/div/div[2]/div[1]/table/tbody'
         element = tree.xpath(xpath)

         pesticide_types = element[0].text_content().strip()
         pesticide_types

Out[25]: 'O0\n\t\t\t\tMITICIDE\n\t\t\t\n\t\t\n\t\t\t\n\t\t\t\tN0\n\t\t\t\tINSECTICIDE'
```

```
In [24]: import re
         re.findall(r'\b[A-Z]{3,}\b', pesticide_types)

Out[24]: ['MITICIDE', 'INSECTICIDE']
```

Problem 2 - Python solution 2 - selectors + BeautifulSoup

Option 2: Use selectors/tags - trickier but better option when we need more elements from different parts of HTML (e.g. all headings). It requires investing HTML structure in more detail

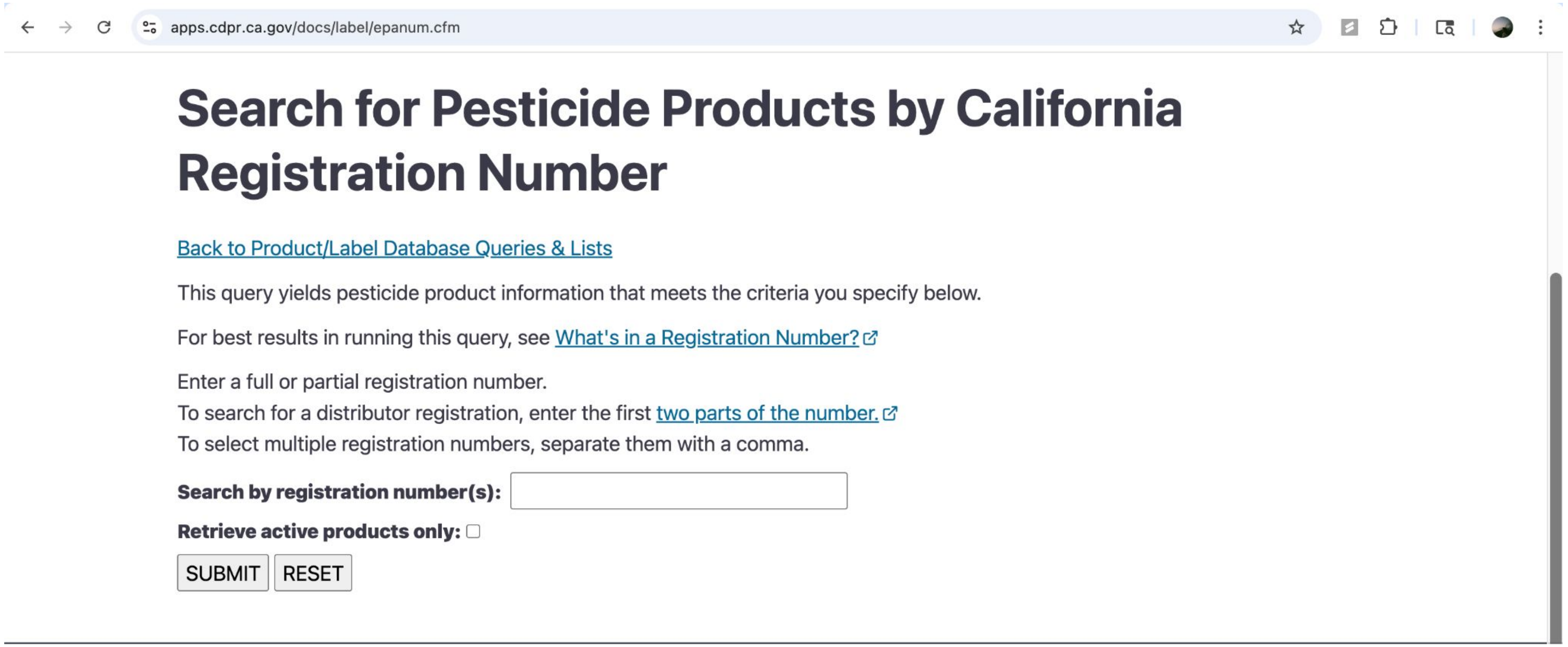
In our case, extract the first table (soup.find(table) command) → extract all elements from the table, choose elements #1 and #3 (Python starts count from 0)



```
In [48]: from bs4 import BeautifulSoup

         url = 'https://apps.cdpr.ca.gov/cgi-bin/label/labrep.pl?fmt=1&63069=on'
         response = requests.get(url)
         soup = BeautifulSoup(response.content, 'html.parser')

         table = soup.find('table') #we are lucky because we need the first table
         entries = table.find_all('td')
         print(entries[1].text.strip())
         print(entries[3].text.strip())

         MITICIDE
         INSECTICIDE
```

Problem 3 - Interact with elements on the website

We need the program to enter “63069” in the bar and press “submit”



Search for Pesticide Products by California Registration Number

Back to Product/Label Database Queries & Lists

This query yields pesticide product information that meets the criteria you specify below.

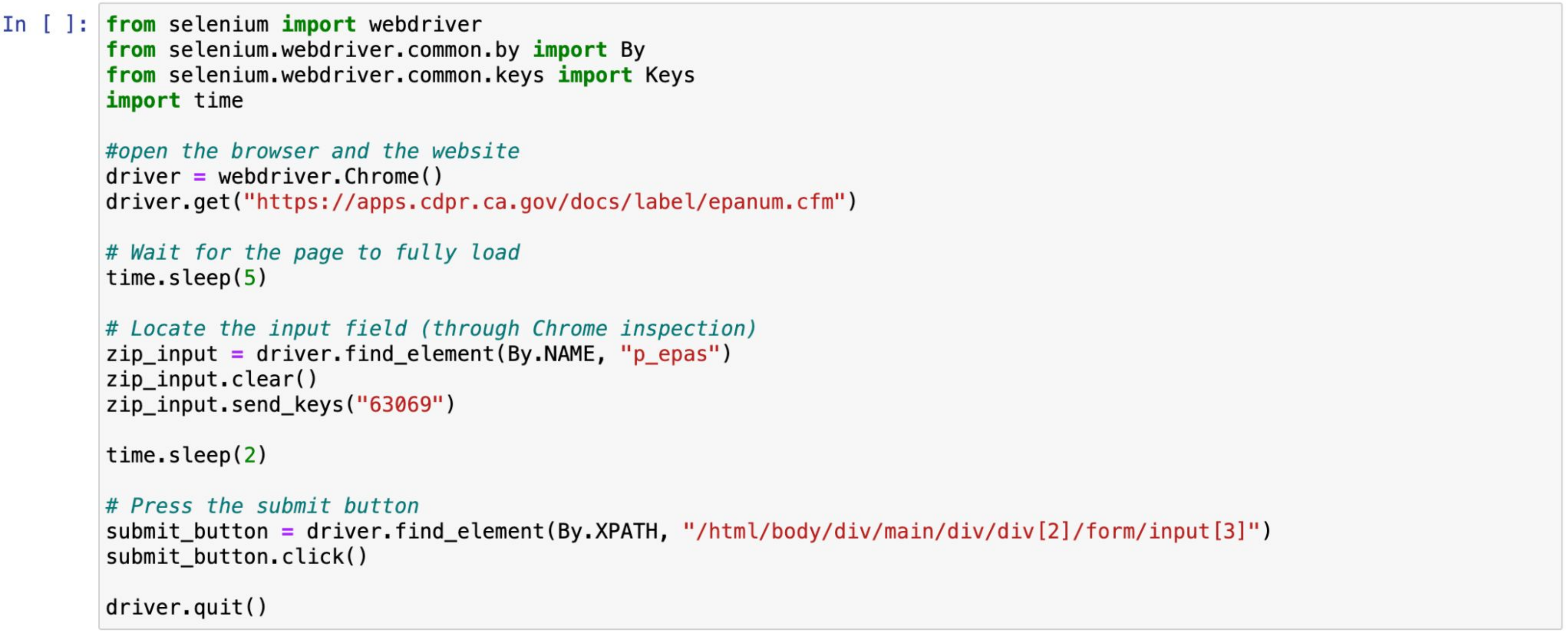
For best results in running this query, see What's in a Registration Number?

Enter a full or partial registration number.
To search for a distributor registration, enter the first two parts of the number.
To select multiple registration numbers, separate them with a comma.

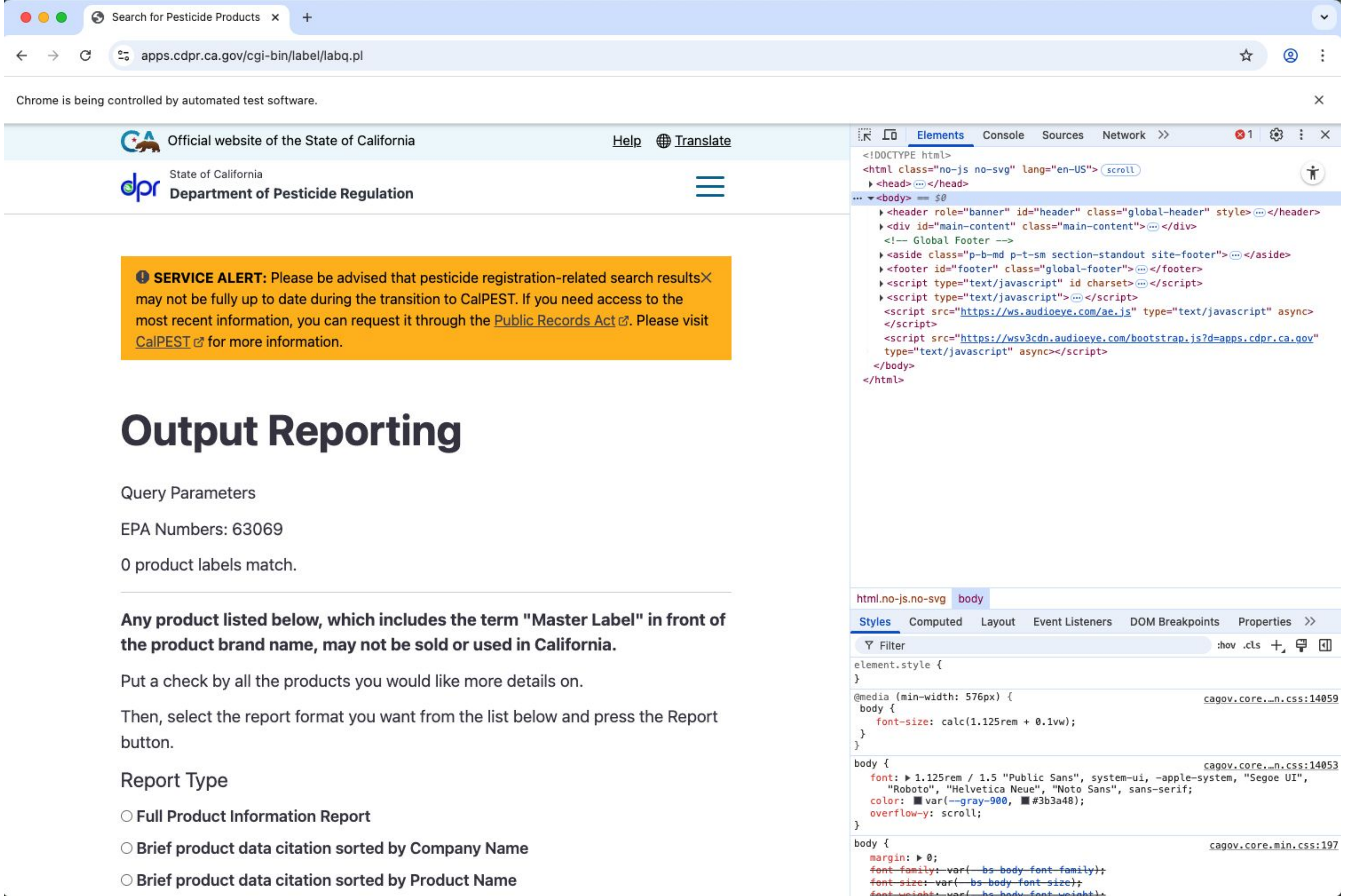
Search by registration number(s):

Retrieve active products only:

Problem 3 - Python solution with Selenium



```
In [ ]: from selenium import webdriver
        from selenium.webdriver.common.by import By
        from selenium.webdriver.common.keys import Keys
        import time

        #open the browser and the website
        driver = webdriver.Chrome()
        driver.get("https://apps.cdpr.ca.gov/docs/label/epanum.cfm")

        # Wait for the page to fully load
        time.sleep(5)

        # Locate the input field (through Chrome inspection)
        zip_input = driver.find_element(By.NAME, "p_epas")
        zip_input.clear()
        zip_input.send_keys("63069")

        time.sleep(2)

        # Press the submit button
        submit_button = driver.find_element(By.XPATH, "/html/body/div/main/div/div[2]/form/input[3]")
        submit_button.click()

        driver.quit()
```


Search for Pesticide Products
apps.cdpr.ca.gov/cgi-bin/label/labq.pl
Chrome is being controlled by automated test software.
Official website of the State of California
Help
Translate
State of California
Department of Pesticide Regulation
Elements
Console
Sources
Network

SERVICE ALERT: Please be advised that pesticide registration-related search results may not be fully up to date during the transition to CalPEST. If you need access to the most recent information, you can request it through the Public Records Act. Please visit CalPEST for more information.

Output Reporting

Query Parameters

EPA Numbers: 63069

0 product labels match.

Any product listed below, which includes the term "Master Label" in front of the product brand name, may not be sold or used in California.

Put a check by all the products you would like more details on.

Then, select the report format you want from the list below and press the Report button.

Report Type

○ **Full Product Information Report**

○ **Brief product data citation sorted by Company Name**

○ **Brief product data citation sorted by Product Name**

```
<!DOCTYPE html>
<html class="no-js no-svg" lang="en-US"> scroll
  <head>…</head>
… <body> == $0
    <header role="banner" id="header" class="global-header" style>…</header>
    <div id="main-content" class="main-content">…</div>
    <!-- Global Footer -->
    <aside class="p-b-md p-t-sm section-standout site-footer">…</aside>
    <footer id="footer" class="global-footer">…</footer>
    <script type="text/javascript" id charset>…</script>
    <script type="text/javascript">…</script>
    <script src="https://ws.audioeye.com/ae.js" type="text/javascript" async>
    </script>
    <script src="https://wsv3cdn.audioeye.com/bootstrap.js?d=apps.cdpr.ca.gov"
    type="text/javascript" async></script>
  </body>
</html>
```

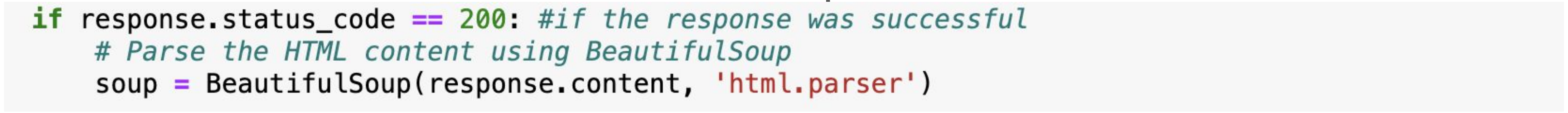
```
html.no-js.no-svg  body
Styles  Computed  Layout  Event Listeners  DOM Breakpoints  Properties
Filter                                                :hov .cls
element.style {
}
@media (min-width: 576px) {                        cagov.core.…n.css:14059
 body {
   font-size: calc(1.125rem + 0.1vw);
 }
}
body {                                             cagov.core.…n.css:14053
   font: 1.125rem / 1.5 "Public Sans", system-ui, -apple-system, "Segoe UI",
      "Roboto", "Helvetica Neue", "Noto Sans", sans-serif;
   color: var(--gray-900, #3b3a48);
   overflow-y: scroll;
}
body {                                             cagov.core.min.css:197
   margin: 0;
   ~~font-family: var(--bs-body-font-family);~~
   ~~font-size: var(--bs-body-font-size);~~
```

Selenium

- Mainly used for web testing.
- You can find elements on the web page by their ID, name, XPath, link, link text, tag name, class name, or CSS selector.
- You can press buttons, fill in forms with text, drag and drop, navigate forward and back, scroll through pages
- You can wait before taking an action until some element fully loads
- Good resource to learn more

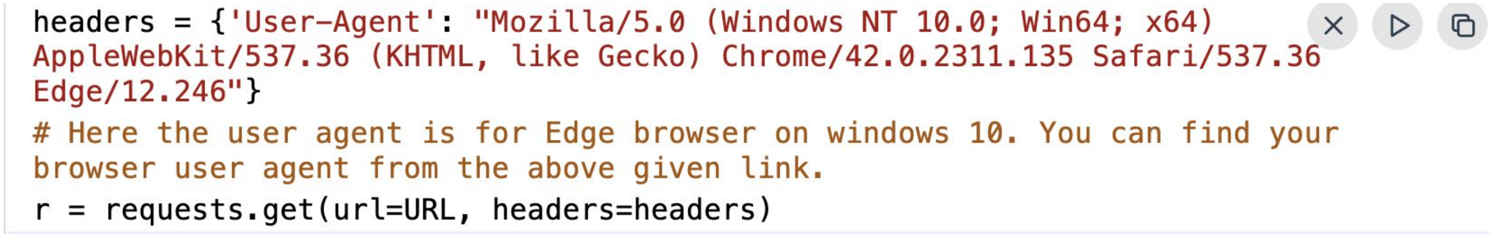
Additional considerations

- Add clauses to handle errors, for example:



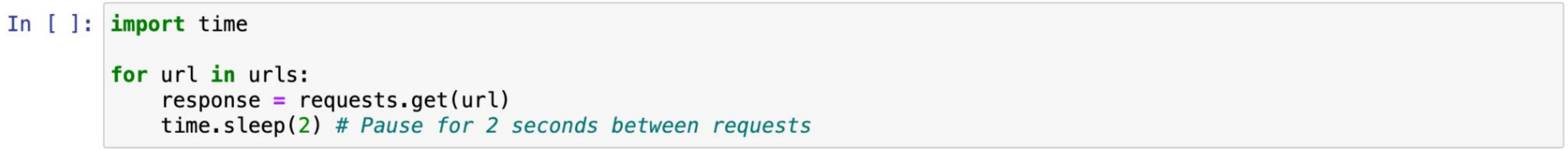
```python
if response.status_code == 200: #if the response was successful
    # Parse the HTML content using BeautifulSoup
    soup = BeautifulSoup(response.content, 'html.parser')
```

- If response.status_code is not 200, it's likely because the server recognizes you as a "bot"
  - Add a browser user agent that mimics the behavior of your browser to your original get request. Find your agent here



```python
headers = {'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge/12.246"}
# Here the user agent is for Edge browser on windows 10. You can find your browser user agent from the above given link.
r = requests.get(url=URL, headers=headers)
```

- With multiple requests, add pauses to your code to avoid getting blocked



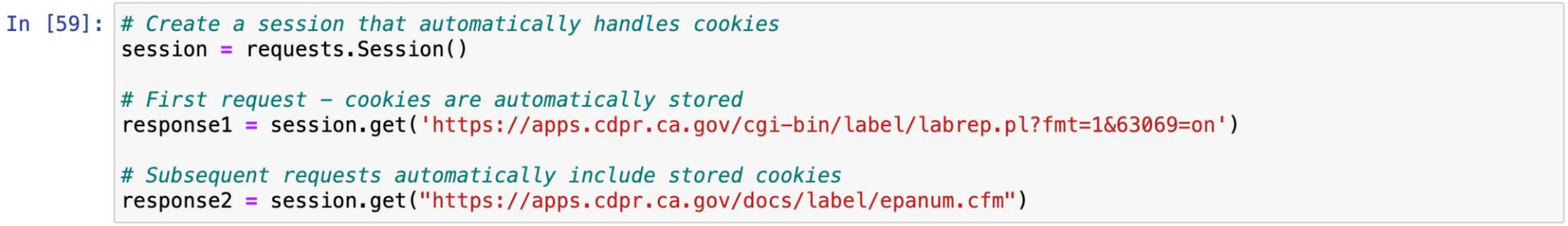
```python
In [ ]: import time

        for url in urls:
            response = requests.get(url)
            time.sleep(2) # Pause for 2 seconds between requests
```

Cookies

You should handle **cookies** in some cases

- They store session info and user preferences
- Websites use them to track your sessions and track bots
- Without proper handling, you may be logged out or see different content than expected
- Setting proper cookies can mimic a real browser

For example, with continuous website browsing, use sessions:



```
In [59]: # Create a session that automatically handles cookies
         session = requests.Session()

         # First request - cookies are automatically stored
         response1 = session.get('https://apps.cdpr.ca.gov/cgi-bin/label/labrep.pl?fmt=1&63069=on')

         # Subsequent requests automatically include stored cookies
         response2 = session.get("https://apps.cdpr.ca.gov/docs/label/epanum.cfm")
```

Read more here

Challenges

- Sometimes, anti-bot protection systems on websites recognize you as a bot even when you add all parameters to mimic regular user behavior
    - May have to limit the number of requests during the day
    - This library may help but use with caution
- Your IP may be blocked by some websites, be careful
- Web scraping can be painfully slow sometimes
    - Can't do much about it but a good thing is it can still run in the background while you are busy with other things
- Websites change more regularly than you'd expect
    - May have to edit your script from time to time
- Web scraping different websites will be different
    - They have different structures
    - Some websites have better anti-bot protection systems
- Web scraping often requires thorough HTML inspection and using some tricks (finding interesting patterns) to help you locate the elements