**Web Scraping – Example 2**

**Author:** Apurva Sachin Kulkarni

**Date:** 19th August, 2025

**Introduction**

This project focuses on web scraping using Selenium in Python to collect End-of-Life (EOL) data for hardware and software products from Palo Alto Networks’ support pages. By automating browser actions, extracting structured data from dynamically generated webpages, and transforming it into clean tabular form, the scripts demonstrate how web scraping can be used for real-world data collection and reporting tasks. The results are saved as CSV files for further analysis or archival purposes.

**Code 1: Hardware EOL Data Scraper**

**Step-by-step explanation**

1. **Import required modules**
   * selenium components (webdriver, Service, Options, By, WebDriverWait) to control Chrome and fetch web elements.
   * pandas for storing and transforming scraped data.
   * webdriver\_manager.chrome for automatically downloading the appropriate ChromeDriver.
2. **Class TableScraper – OOP approach**
   * **\_\_init\_\_ method:**
     + Initializes Chrome in either **headless** mode or visible mode.
     + Maximizes or minimizes windows as required.
     + Sets a **WebDriverWait** timeout (15 seconds by default) for waiting on elements.
   * **open(url) method:** Opens the target webpage.
   * **minimize() method:** Minimizes the browser explicitly (useful when running non-headless).
   * **close() method:** Properly quits the browser after scraping.
   * **fetch\_table\_rows(xpath) method:**
     + Locates table rows using an XPath selector.
     + Extracts both <th> (headers) and <td> (data) cells from each row.
     + Returns data as a list of lists.
3. **Main program execution**
   * Instantiates TableScraper.
   * Opens the Palo Alto **hardware EOL page**.
   * Extracts the HTML table rows using the XPath './/table//tr'.
   * Closes the browser.
   * Converts the raw list into a **DataFrame**:
     + Uses the first row as column headers (data[0]).
     + Drops irrelevant columns (End-of-Sale Date, Last Supported OS).
     + Renames columns to meaningful names (productName, EOL Date).
     + Converts EOL Date to yyyy/mm/dd format using pd.to\_datetime.
   * Saves cleaned data as **Hardware EOL Dates.csv.**

**Code 2: Software EOL Data Scraper**

**Step-by-step explanation**

1. **Import required modules**
   * Selenium components (webdriver, By, WebDriverWait, expected\_conditions) for automated browser navigation.
   * pandas for data storage.
   * datetime for parsing different date formats.
   * time for adding explicit wait delays where needed.
2. **normalize\_date(text) function**
   * Cleans up date strings.
   * Converts various formats ("March 5, 2021", "Mar 2021", "2021-03-05") into standard **yyyy-mm-dd**.
   * Returns placeholders unchanged if no valid date is found (like "TBD", "N/A", "-").
3. **Browser setup and navigation**
   * Launches Chrome and navigates to Palo Alto’s **EOL summary page**.
   * Uses time.sleep(5) to allow React components to load.
   * Scrolls gradually (window.scrollTo) to trigger lazy loading of tables.
   * Waits until at least one data row appears using WebDriverWait with EC.presence\_of\_element\_located.
4. **Data extraction**
   * Locates **all tables containing “Version” in their header** using XPath.
   * For each table:
     + Assigns a software name (software\_name="Prisma Access Browser" in this script; in a general scraper this would be dynamically detected).
     + Iterates through each row to extract:
       - **Version** (first column)
       - **Release Date** (second column, normalized)
       - **EOL Date** (third column, normalized)
     + Adds the row to a records list.
5. **Final processing**
   * Quits the browser to free resources.
   * Converts the records into a **DataFrame** with columns:
     + Software Name, Version, EOL Date, Release Date.
   * Saves cleaned data as **Software EOL Dates.csv**.

**Conclusion**

Both scripts successfully demonstrate practical web scraping techniques using Python and Selenium. The first script applies object-oriented programming (OOP) for maintainable, reusable code to handle structured hardware EOL tables, while the second script manages dynamic and irregular software data with date normalization and lazy-loading support. Together, they showcase how automated data pipelines can convert complex, real-time web data into clean CSV datasets suitable for analytics, reporting, or archival use.

\*\*\*