Task 2 Read Me:

Files Included:

- Task 2 (1): Contains the code to scrape all list and product pages except for the GAP list page. To run the code, you will need to install three libraries: requests, bs4, and BeautifulSoup. The output will be saved in an Excel file named "Meman_diaby_task_2_1.csv". The code will print a confirmation message upon successful execution. Note that scraping the GAP list page will fail.

- Task 2 (GAP List Page): Contains the code necessary to scrape GAP list page data. To run this code, you'll need the Panda and Selenium libraries. Additionally, you'll need to download Chromedriver. Selenium WebDriver will interact with the webpage to close the promotional banner that may interfere with scraping. The output will be saved in an Excel file named " gap_products.xlsx".

Approach:

1. Scraping List Pages:

- Utilizes BeautifulSoup library to parse HTML content.

- Extracts relevant information such as product name, price, and discounted price (if available) from each list page URL.

- Considers the first 10 products on each list page.

2. Scraping Product Pages:

- Extracts additional information such as brand, number of photos, number of colors, and product description from each product page URL.

- Uses different CSS selectors or XPath expressions to target specific elements containing the required information.

3. Handling Promotional Banner (GAP List Page):

- Utilizes Selenium WebDriver to interact with the webpage and close the promotional banner if it appears.

- Identifies the banner by its XPath expression and closes it programmatically.

4. Scrolling and Loading More Products (GAP Website):

- Ensures all products are loaded on the GAP list page by scrolling down to the bottom of the page.

- Dynamically scrolls using Selenium WebDriver and implements a loop to detect the end of the page.

- Adjusts time intervals between scrolls for proper loading of products.

How to Use Selenium WebDriver:

To effectively utilize Selenium for web scraping tasks, ensure you have the ChromeDriver executable installed on your system. This facilitates communication between your Python script and the Chrome browser. You can download the ChromeDriver executable from the official website or use WebDriver Manager, which automates this process. Once installed, initialize the WebDriver, navigate to the desired URL, interact with page elements, handle page loading delays, and implement robust exception handling. Finally, execute your Python script to run the Selenium-driven web scraping tasks smoothly.

Notes:

1. Promotional Banner (GAP list page):

- The GAP website displays a promotional banner that may obstruct the scraping process.

- Selenium WebDriver is employed to close the banner automatically if it appears.

2. Scrolling and Loading Products (GAP list page):

- Scrolling down to load more products is necessary due to potential dynamic loading on the GAP list page.

- A loop is utilized to continuously scroll down until the bottom of the page is reached.

- Time intervals between scrolls are adjusted to allow for proper loading of products.

3. Data Storage and Export:

- Scraped data is stored in a structured format for further analysis.

- The pandas library is used to convert the scraped data into a DataFrame.

- The DataFrame is exported to an Excel file for easy access and sharing.

4. CSS Selectors and XPath Expressions:

- Accurate CSS selectors and XPath expressions are employed to target specific HTML elements containing the required information.

- Exception handling is implemented to handle cases where elements cannot be found or contain unexpected data.