**Web scrapping -1¶**

**Task description**

You will crawl over the input list to get information (attributes) for each product of the list. The script needs to correctly reads the input, do the scraping task for each link, and export the result as excel with the expected format (see Expected result below).

The code will be written in Python and you are free to use the IDE of your choice and all Python libraries you need to complete the task. The performance of the solution, as well as the documentation and understandability of the code will be evaluated.

**Expected result**

The solution needs to be a zipped folder named “Test\_Coding\_yourname\_dd.mm.yyyy.zip” containing:

• The input file “input.xlsx”

• The Python script “test\_coding.py” containing the code to execute the task

• An excel file “Webshop\_ProdData\_ dd.mm.yyyy-hhmmss.xlsx” containing the product data (generated by your script)

* Attributes for each product:
  + url: link of the product (same as the input)
  + product\_title: title displayed on the product page
  + variation\_id: unique identifier used to differentiate different children’s variations of one parent product (e.g.: product X available in Y different sizes = parent X has Y variations and therefore Y unique variation IDs)
  + size: size of the product
  + price: price of the product
  + ean: European Article Number (barcode), standard number of 13 digits
  + img1\_url: url of the 1st picture visible on the product page
  + img2\_url: url of the 2nd picture visible on the product page (in the carrousel)

N.B: the images urls can differ depending on the method you used but it will be considered as correct as long as it refers to the correct picture (see example bellow with 2 different url but same picture).

**Objectives**

1. **Scrape Product Information**: Extract detailed product information from a set of given URLs. The information includes product title, variant ID, size, price, EAN (European Article Number), and image URLs.
2. **Handle HTML Structure**: Accurately parse the HTML structure of each product page to locate and extract the necessary data, ensuring robustness against variations in the HTML.
3. **Data Aggregation**: Organize the extracted data into a structured format (e.g., a Pandas DataFrame) for easy analysis and potential further processing.
4. **Error Handling and Robustness**: Implement error handling to manage potential issues such as missing data, incorrect HTML structure, or network-related errors.

**Results**

1. **Product Data Extraction**:
   * Successfully extracted product titles, variant IDs, sizes, prices, EANs, and image URLs from the given product pages.
   * Utilized BeautifulSoup to navigate the HTML and extract the necessary information.
2. **HTML Parsing**:
   * Implemented functions to handle the extraction of specific data points, such as product titles, sizes, prices, and image URLs.
   * Addressed potential issues with missing or malformed HTML elements through error handling.
3. **Data Aggregation**:
   * Collected the extracted data into a dictionary and then converted it into a Pandas DataFrame for better organization and analysis.
   * Ensured that the DataFrame contains all the required fields with appropriate data.
4. **Error Handling**:
   * Incorporated try-except blocks to manage exceptions during data extraction.
   * Printed debug information to identify and troubleshoot issues during the scraping process.

**Example DataFrame Output**

Here's an example of what the final DataFrame might look like:

| **url** | **Product\_title** | **Variation\_id** | **Size** | **Price** | **Ean** | **Img1\_url** | **Img2\_url** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| https://example.com/page-1 | Ganzjahresbettdecke | 30732042993796 | 200x220 | 2290 | 4063585977446 | https://example.com/image1\_1.jpg | https://example.com/image1\_2.jpg |
| https://example.com/page-2 | Another Product | 30732042961028 | 155x220 | 1990 | 4063585977453 | https://example.com/image2\_1.jpg | https://example.com/image2\_2.jpg |
| https://example.com/page-3 | Yet Another Product | 39980492161156 | 160x220 | 1990 | 4063585934807 | https://example.com/image3\_1.jpg | https://example.com/image3\_2.jpg |

**Steps Taken**

1. **Session Setup**:
   * Initialized a requests.Session with appropriate headers to mimic a real browser.
2. **Data Extraction Functions**:
   * Defined functions for extracting specific data points from the HTML using BeautifulSoup.
3. **Loop Through URLs**:
   * Iterated over the list of product URLs, made HTTP requests, and used the extraction functions to gather the required data.
4. **Error Handling**:
   * Used try-except blocks to catch and manage errors, ensuring the script continues to run even if it encounters issues with some URLs.
5. **Data Storage**:
   * Stored the extracted data in a dictionary and then converted it into a Pandas DataFrame for better organization.

By following these steps and meeting the objectives, the task of scraping product information and organizing it into a structured format was accomplished effectively.

**Key Points:**

1. **Initialization and Session Setup**:
   * Ensure the requests.Session is properly initialized with headers to mimic a real browser request.
2. **Functions**:
   * extract\_json\_data\_ean(soup): Extracts JSON data embedded in the HTML.
   * images\_list(soup): Extracts image URLs.
   * prod\_names(soup): Extracts product names.
   * sizes(soup): Extracts chosen size.
   * vpb\_lists(soup): Extracts variant ID, price, and barcode based on the given price.
3. **Data Collection Loop**:
   * Iterates over the list of URLs, processes each page, extracts required data, and appends it to the data dictionary.
   * Includes error handling and prints debug information to help identify issues.
   * Sleeps for 2 seconds between requests to avoid getting blocked.
4. **Output**:
   * Creates a DataFrame from the collected data and prints it.

Run this script in a controlled environment to ensure it works as expected. Make sure the product page URLs are valid and accessible.