Final Home Assignment:

Maor Nave – ID.313603391, Tel – 0543121144

Date: 17/08/2023

**Code Guide: IEEE Crawler programming, Data Processing and Analysis**

The code featured in this research output embodies a crafted program designed to construct a web crawler tailored for a specific online platform - the IEEE academic journal. This crawler is engineered to undertake targeted data extraction operations, responding to stipulated research requisites. It subsequently generates an Excel-format data file as output, wherein each entry encapsulates pivotal information distilled from individual articles.

Integral to this endeavor is an elegantly structured output repository. This repository systematically compiles pertinent image data from articles across distinct annual epochs within the broader article chronology. The nomenclature of each repository folder precisely mirrors the appellation of the pertinent article in its corresponding year and decade, meticulously aligned with the journal's edition issue.

1. Key Considerations:
2. The foundational construct of the crawler as predicated upon the credentials of a student or academic staff user at Bar Ilan. This authentication is orchestrated via interfacing with the Windows Authenticator application.
3. The mechanics of the crawler is orchestrated within the ambit of the Chrome browser.
4. The workflow necessitates the integration of Google's webdriver plugin.
5. Ensuring seamless execution of the code mandates the prerequisite installation of relevant python libraries which will be presented later on this work.
6. It is recommended to activate the code with screen shutdown as Never.
7. An active connection to the Internet before starting the program run.
8. Installation Procedures:
9. Windows Authenticator:
   1. Download the application from the provided link: <https://www.microsoft.com/en-us/security/mobile-authenticator-app>.
   2. Post-installation, populate requisite details to establish a linkage with the new authenticator, employing university system credentials.
   3. After enrolling a new authenticator, if applicable, input the provided code to establish connection with the pertinent portal.
10. Chrome Browser:
    1. Acquire the browser by downloading and installing it from the following URL: <https://www.google.com/chrome/?brand=FHFK&gclid=Cj0KCQjwoeemBhCfARIsADR2QCvT7oiNy149juh-nchVKX_Ev2sW6eYSRspa9S2_WovFEFFt1gI102QaArn0EALw_wcB&gclsrc=aw.ds>.
11. Chrome WebDriver:
    1. Access the latest version of the Chrome WebDriver through the download link: <https://chromedriver.chromium.org/downloads>.
    2. Transfer the downloaded files (post-extraction from the compressed format) to the directory housing the browser executable. Typically, this location corresponds to "Program Files/Chrome".
12. Configuration File:

For facilitating an expedient user experience, an external configuration file has been devised, bifurcating work into four discrete segments:

1. University Systems Credentials - Furnishes the means to log into Windows-based systems.
2. Browser Paths - Provides a central repository for browser-related parameters.
3. Code Execution Parameters - Consolidates essential parameters governing code execution.
4. XPath Paths - Catalogs XPath paths, essential for crawler-based data extraction.
5. Folder paths – all generic folders that are generated while the program run to save data.
6. Libraries:
7. Numpy (np): Provides functions for working with arrays and mathematical computing.
8. pandas (pd): Offers data manipulation and analysis tools using DataFrames.
9. yaml: Provides methods for working with Yaml data.
10. Selenium: import webdriver ,service, Keys, By, WebDriverWait, EC, to interact with web browsers.
11. Time: to help loading web pages throughout the code.
12. Genderize: API request tool for determine names gander.
13. OS: facilitates seamless programming involving file paths.
14. Pickle: Facilitates convenient serialization of data into multiple layers of nested dictionary-like structures.
15. Urllib.request: Offers straightforward methods for accessing visual content such as images and photographs through hyperlinks, enabling their preservation as files on a computer.
16. Re: Offers modules dedicated to the task of string cleansing and manipulation.
17. Keyboard: Provides a means to manage keyboard input, mitigating potential issues leading to inadvertent screen shutdowns.
18. DateTime: Utilized for the conversion of publication dates of individual articles, enhancing accuracy and consistency across the dataset.
19. Functions:
20. General Functions:
21. call\_function\_with\_retry(function, max\_retries, article\_path=False):

* This function facilitates the iterative execution of bot functions. It encompasses various scenarios and provides options to condense the entire program for any accepted function. The function takes into account the count of prior attempts required for invoking its functions, with this process being regulated by the configuration file set at the initiation of the task.

1. call\_driver():

* This function serves a dual purpose: it both initializes the Chrome driver object and establishes the driver's element wait settings. Subsequently, it establishes the genderize object to facilitate access to the API for determining the genders of

writers.

1. press\_key():

* This function, upon each invocation, engages the keyboard library to simulate a key press action, ensuring that the screen remains active while the program is running, thereby preventing unintended screen shutdowns.

1. folder\_exists(folder\_name):

* This function accepts the name of a folder intended for testing purposes. If the specified folder exists on the computer, the function provides a notification to reflect its presence. In case the folder is not found, the function creates a new folder within the appropriate directory structure aligned with the program's scope and subsequently communicates this action.

1. yaml\_loader(path):

* reads yamls files from paths.

1. yaml\_dumper(path, data):

* write yaml files to path.

1. pickle\_loader(path):

* reads pickle files from paths.

1. pickle\_dumper(path, data):

* write pickle files to path.

1. find\_element\_xpath\_clear(path):

* A function that leverages the Chrome driver's element waiting capabilities to execute the process of sanitizing and inputting strings into login fields within the Windows authenticator.

1. find\_element\_click(method, path):

* A function that leverages the Chrome driver's element waiting capabilities to execute the process of clicking on the webpage.

1. full\_articles\_links\_list\_creator(rev\_articles\_links\_paths\_list):

* Since data pertaining to links for each article within the journal is provided based on yearly issuance, with the aim of establishing accessible exit points in case of program crashes, the function essentially aggregates all the data acquired through the "crawl\_journal" function. It consolidates this information into a unified record encompassing links from all articles.

1. full\_names\_checker(rev\_authors\_list):

* A function that assesses an article's author name, and if the name appears duplicated, it avoids accessing the API. This strategy is employed to conserve the quota of requests that can be made per day.

1. text\_cleaner(text):

* This function is designed to carry out a refinement process for every string that enters the dictionary, containing information pertinent to a specific article. The cleaning process involves the removal of characters from a predefined list, which includes symbols like '$', '^', '\*', '/', '@', '#', '\_', '\', '{', '}', as well as the newline character '\n'.
* Additionally, a choice has been made to eliminate consecutive spaces. However, it's important to note that Unicode characters are retained, as they accurately represent the intended character and are typically preserved in most basic interpreters, allowing the data to remain in its original form.

1. articels\_list\_checker(year):

* The function's purpose is to verify the presence of a file within the "articles\_lists\_by\_rev\_year" folder. This file is responsible for storing URLs obtained from the "crawl\_journal" function, among other data. The function operates under certain assumptions: firstly, that these URLs encompass all relevant issues from 2015 to 2023; secondly, that websites saved in the list are not cover pages.
* This determination is based on the convention that cover papers lack page links. However, it has been observed that exceptions exist where cover papers possess links, which are handled in the "extract\_article\_info" function. When this file is identified, the function triggers an alternative search mechanism within the "crawl\_journal" function.

1. jpg\_writer(image\_url, output\_path):

* The function's role is to access the URL derived from the "figures\_data\_extractor" function. This URL retrieval is related to the extraction of information about each figure contained within an article. Subsequently, the function proceeds to extract the image contained in this URL, saving it in a standardized JPG format with dimensions of 600 by 800 pixels.

1. Semi-Core functions:
2. fill\_full\_article\_data\_dict(input\_df, general\_dict, authors\_dict, figures\_dict, tables\_dict):

* The function accepts the projected output configuration and generates a dictionary from it. This resultant dictionary will then be stored by the "extract\_article\_info" function every ten iterations.
* Additionally, the function is designed to receive dictionaries from the "extract\_article\_info" function. These dictionaries pertain to tables, pictures (figures), and authors within each article.

1. get\_gender(name):

* The function essentially initiates a request to the API server with the purpose of obtaining the gender probability for a given writer's name.
* Furthermore, the function considers the API's daily limit of 1000 name requests and accordingly preserves a "names\_dict" pickle file. This file contains a record of all name tests previously dispatched to the API endpoint.
* Simultaneously, the function accounts for variations in first name placement across different contexts. The function generates requests for all plausible combinations of the name by segmenting the name string using spaces. The assessment of the credibility of a given first name hinges on two factors: the frequency of searches performed within the API's information repository for that specific name (count), and the probability of alternative explanations for the name.This function is triggered by the "authors\_data\_extractor" function.

1. none\_figuers\_dict\_extractor():

* Function that returns an empty dictionary for figures data in extreme cases of no figures found on the figures div or in a case that a figures div is not found.
* The function keeps on not available data conventions defined on the assignment.
* The function is initiated by figuers\_data\_extractor function.

1. figuers\_data\_extractor(figures\_bar\_div,data\_year,paper\_name):

* This function specializes in extracting image data of figures embedded within the article itself. Its invocation takes place through the "extract\_article\_info" function. The function receives the "figures\_bar\_div," which serves as the location housing the pertinent information about the article's images.
* The function can deal with another extreme case, a div value of figures exist with no figures inside, will return empty figures dictionary with the relevant conventions for no available figures.
* Concurrently, it takes in the year of the article's release to facilitate precise local storage adjustments for the figures.
* Simultaneously, the function is supplied with the article's name, aiding in the configuration of the saving path for website images. The naming conventions for final folders (derived from the article name) are confined to a maximum of 80 characters. Any spaces within the path are substituted with underscores, mitigating potential crashes arising from character limits due to lengthy article names.
* The output of the function is a dictionary encompassing the details of the first ten figures within the article. The function accommodates various scenarios, generating an appropriate dictionary even when there are no figures on the site or when there are more than ten figures, capturing adjusted information regarding the total figure count.
* To ensure accurate extraction of figure information from the article, the function employs a tailored while loop coupled with a wait mechanism. This approach is designed to capture the relevant div element with the appropriate class, containing the figures details, and with that in a case of logout the function will log you in on the while loop (as extreme use case to help the program continuation). Such an approach effectively addresses edge cases where the program had previously encountered issues. The utilization of this method reduces reliance on the sleep function, contributing to more efficient execution.

1. tabels\_data\_extractor(full\_text\_section\_div):

* This function is dedicated to extracting the existing table data from a specific article page. Its activation is also triggered by the "extract\_article\_info" function. When invoked, the function is provided with a segment of the webpage containing comprehensive details about the article itself. This page is divided into various sections, all encapsulated within a prominent div, essentially forming the article's body.
* To successfully extract table information from the article, a targeted search is executed for a div with a specific ID that corresponds to the table names, such as "Table 1," "Table 2," and so on. By pinpointing these divs, we can navigate within the overarching div that holds the entire article's content. This enables us to extract the text describing each table, as well as access all tables present within the article.
* The function has the capability to handle scenarios where tables are absent in the article, and it also accounts for cases where there are more than ten tables.
* The function outputs a data dictionary.

1. authors\_data\_extractor(authors\_bar\_div):

* This function serves the purpose of extracting author data from each article, aligning with the specified job requirements. Specifically, it focuses on retaining information concerning the first and last authors exclusively.
* The information saved encompasses the names of both the first and last authors, along with their gender probability and determined gender (determined via the "get\_gender" function). Additionally, the total count of authors featured in the article is recorded.
* The function is initiated through the "extract\_article\_info" function. When called, it receives data in the form of a div element containing comprehensive details about all the authors associated with the article.
* The function adeptly handles exceptional situations, including cases where the first and last authors are the same individual—usually indicating a single-author article. It also addresses scenarios where images (photos) are present, but no authors are listed. While this is typically indicative of cover articles, the initial assumption is that an absence of author links indicates an illegitimate article. However, due to a rare instance where author names were not specified, a decision was made to provide an organized blank information response in such cases.
* The function outputs a data dictionary.

1. publication\_doi\_div\_exctractor():

* This function is designed to patiently await the identification of specific divs containing pertinent information within the article. These divs specifically hold the article's DOI (Digital Object Identifier) and its corresponding release date.
* Through repeated runs, it was observed that these divs occasionally exhibited a loading time longer than the average. In response, a decision was made to incorporate a loop that remains active until these divs present themselves with the accurate information.
* The activation of this function takes place within the "extract\_article\_info" function. This arrangement is established to facilitate the generation of a comprehensive dictionary. This dictionary is subsequently integrated with other dictionaries, such as those pertaining to authors, images (figures), and tables. These dictionaries are generated during the core process within the "extract\_article\_info" function.

1. cover\_articles\_deletor(article\_link, full\_articles\_links\_list, general\_article\_data\_dict):

* This function becomes operational in cases where the "extract\_article\_info" function encounters a failure, primarily when identifying an article page as an internal cover article. This scenario was detailed in Section 2 of the "articels\_list\_checker" function documentation.
* As outlined, the conventional method to identify a cover paper is based on the absence of a link to its own page when the "crawl\_journal" function is invoked. However, program execution revealed instances where articles categorized as cover papers did not adhere to this described convention. Consequently, a revised convention was necessary for determining articles as cover papers, especially when loaded via the "extract\_article\_info" function.
* When a program crash occurs via this function, an argument is passed to ascertain if an article is potentially a cover paper. This is achieved by checking for the non-appearance of a relevant div related to authors, and further verifying the presence of elements characterizing the article's body. In cases where an article is suspected to be a cover paper, the function excludes it from the entire list. This list is then saved for subsequent loading within the "extract\_article\_info" function. Additionally, the function restarts work from the correct index where the previous run failed. The identified cover papers are consolidated into a new list following the revised convention.

1. Full-Core functions:
2. build\_full\_data\_df(rev\_articles\_data\_paths\_list):

* This function comes into play once the "extract\_article\_info" function completes its execution. Its primary role is to gather and consolidate all the data generated by the "extract\_article\_info" function. This data is organized into ten dictionaries list, each dictionary encompassing all the essential information required for a single entry within the final DataFrame that we intend to save.
* The function then carries out iterations equivalent to the number of files for which information is available. Subsequently, it compiles these iterations into a final DataFrame, which is then exported to an output file named "output.xlsx."

1. crawl\_login():

* This function is triggered by a configuration file toggle, facilitating the reinitialization of the driver and establishing an initial connection to the Journal website via Windows Authenticator. The function receives pertinent data from the configuration file, specifically the username and password associated with the academic institution (Bar Ilan).
* The function executes iteratively as per the number of attempts designated by the operator, which can be modified by adjusting the "max\_retries" parameter within the configuration file. An additional toggle named "activation\_stay\_key" is available for user selection. This toggle dictates whether a prolonged or abbreviated connection should be maintained. This decision hinges on the choice to either press the "remember me" or "forget me" button presented during the connection process.
* At its core, the entire procedure centers around establishing an uninterrupted connection via the function call. The toggles and supplementary capabilities serve as tools for system testing and addressing potential issues stemming from internet disconnections or interactions with the Journal website.

1. crawl\_journal():

* This function initiates an initial crawl of the Journal website, aiming to discover all link addresses within the relevant issues spanning the years 2015 to 2023. These years are specifically chosen as they represent periods where article information is predominantly accessible through browser browsing and not necessarily external files.
* **NOTE** – the papers are published on journals between 2015-2023, but , it doesn’t mean that the paper published on the same year – it can be published before its actual publication on the journal.
* The function's exploration adheres to the convention that each article div is accompanied by a "tag of a" that defines the article's page address. Notably, there are exceptions like cover papers that possess links to their respective pages. Such cases are handled by the subsequent "extract\_article\_info" function.
* In total , 3100 relevant papers were collected.
* Following the accumulation of articles across all issues within a specific year, the function assembles and stores a list of addresses in a designated folder.
* Activation of this function also hinges on a toggle that evaluates whether data re-saving for the relevant years is necessary or whether transitioning to the subsequent "extract\_article\_info" function, responsible for individual article data collection, is more appropriate. Additionally, the function is invoked as many times as deemed relevant for login attempts, guided by the "max\_retries" argument found within the configuration file.

1. extract\_article\_info(article\_path):

* This function is triggered by a toggle, which evaluates whether it's necessary to re-save relevant data (lists of dictionaries stored in an organized folder) or whether it's time to transition to the next function, "build\_full\_data\_df." The latter function manages data collection and the production of the final output file.
* Additionally, the function is invoked a specific number of times, determined by the allowed number of attempts for it to run without crashing. This limit is influenced by potential crash situations, which will be described in this paragraph. The function's execution is also influenced by the "max\_retries" argument present in the configuration file.
* The function receives a list of links named "full\_links\_articles\_list," encompassing various articles published between 2015 and 2023. It initially checks for a more recent version of the file containing links to articles. If found, it loads this updated version. This provision is essential because the function addresses cover papers, which may possess links to article websites and could lead to an initial program crash.
* Moreover, the function operates based on the "articels\_data\_by\_rev\_year" folder and the amount of article data that has been previously generated. It takes into account the index, as information is output every ten articles. During loading, it determines the relevant index for subsequent searches. This optimizes runtime and ensures the preservation of past information.
* The function initializes an argument for identifying articles as covers, and this process employs the "cover\_articles\_deletor" function, as outlined in this work. It generates various arguments: "general\_article\_data\_dict," "authors\_data\_dict," "figures\_data\_dict," and "tables\_data\_dict." These arguments collectively contribute to crafting new information records—complete dictionaries. These dictionaries are stored in the list of comprehensive dictionaries, which describe issued information every ten entries, and are saved in the designated folder.
* figuers\_data\_extractor function helps to ensure that on second case of program failure the website will logon again on its own (for long program runtimes – overnight cases).
* Data generation is orchestrated by initiating each of the Semi-Core functions.
* Due to the function ability to filter all inner cover papers the full output that is inferred by the function will be about ~3000 papers.

1. Main:
2. The function imports the Python document titled "ActivationFunctions," which encompasses all the activation-related functions.
3. It proceeds to load the global variable "data\_config," effectively transferring all configuration file values into the function.
4. The function operates based on a toggle and an activation order for three key functions: "crawl\_login," "crawl\_journal," and "extract\_article\_info."
5. For the "extract\_article\_info" function, it generates the comprehensive list of articles, designated as "full\_links\_articles\_list."
6. Ultimately, the function concludes by generating the final Excel output, encompassing all processed data.
7. Project Folders:
8. input 🡪 stores output data example file named output.xlsx and configuration file.
9. full\_output 🡪 stores the crawler program data output named output.xlsx.
10. articels\_lists\_by\_rev\_year 🡪 stores the lists of articles links until each relevant year.
11. articels\_data\_by\_rev\_year 🡪 stores the lists of articles data per 10 for each file , and with final work index as name.
12. articels\_figures\_by\_rev\_year🡪 stores the figures as 600\*800 pixels jpg format file, per relevant year and paper name.
13. cover\_list 🡪 stores a list of all links to cover articles found with link.
14. full\_articles\_list\_copy🡪 stores a list of all articles links after popping the last cover link.
15. names\_dict 🡪 stores a list of dictionaries with API data request to all names that covered once by the crawler.
16. Code\_guide.docx 🡪 this file.
17. Full\_Code.docx 🡪 Full code in doc file.
18. Main.py 🡪 main python function file.
19. ActivationFunctions.py 🡪 all crawler program functions file.
20. Bibliography:
21. <https://pandas.pydata.org/>
22. <https://selenium-python.readthedocs.io/index.html>
23. <https://docs.python.org/3/library/os.html>
24. <https://pillow.readthedocs.io/en/latest/reference/Image.html>
25. <https://pyyaml.org/wiki/PyYAMLDocumentation>
26. <https://docs.python.org/3/library/re.html>
27. <https://docs.python.org/3/library/pickle.html>
28. <https://genderize.io/>
29. <https://chromedriver.chromium.org/downloads>
30. <https://pypi.org/project/keyboard/>