Part 1 - Web Crawler

**Summary:**

The web-crawler part of the project starts with a single link provided by the user and gathers several other links related to that topic and scrapes information from those websites. The scraped data is then stored into several text files which are used to create the knowledge base.

**Program Flow:**

* Start URL: The program begins with a start URL of Keanu Reeves’ Wikipedia page.
  + It parses through the website and collects all URL’s present in it.
  + It does not include certain URL’s containing phrases which are hardcoded such as ‘%’, ‘wiki’ (to avoid same Wikipedia page in different languages, and ‘donate’.
* URL Collection: It collects related URLs from the start URL and subsequent URLs.
  + All the URL’s are added to the local list variable.
* URL Filtering: Filters URLs to include only those related to Keanu Reeves.
  + All the unnecessary URL’s are filtered by length if length is lesser than 20.
  + Additionally, they should contain either ‘keanu’ or ‘reeves’ in it.
* Data Scraping: Scrapes data from the final list of URLs.
  + Using requests and BeautifulSoup libraries, data which is visible is collected from the websites.
  + All this data are stored in the text files.
  + The first corpus contains the table data of movies and tv shows from the Wikipedia page.
  + The second corpus contains the Wikipedia page data of the subject.
  + Rest of the corpuses are non- Wikipedia page data- articles, newsletters and such.
* Data Cleaning: Cleans the scraped data.
  + The clean\_text() function does a basic clean on the text files to remove unnecessary spaces, non-alphanumeric characters, references (‘[29]’), and tag names.
* Data Filtering: Further filters the cleaned data to remove irrelevant information.
  + The filter\_text() function does the second level of filtering to the text files based on the user’s ban list. To eliminate lines like ‘contact us for more’, ‘read more’ etc.
* Data Storage: For every link parsed, the data scraped is stored in a text file.
  + The filenames are in incremental fashion, so counters are used to traverse through.
* **Knowledge base creation:**
  + The first corpus file – corpus0.txt contains the movies and tv shows tabled data. This file is used to create pickle dictionaries in the knowledge\_base.py. The key is the year number, and the values is the list of movies or tv shows from that year.
  + From the second corpus files, all the text is stored into a dictionary with the
  + doc numbers as key and list of sentence tokenized strings as the values.

Ex: { ‘doc1’ : [‘sentence 1’, ‘sentence 2’ , ‘sentence 3’ ……. ‘Sentence n’],

‘doc2’ : [‘sentence 1’, ‘sentence 2’ , ‘sentence 3’ ……. ‘Sentence n’],

.

.

.

‘doc20’ : [‘sentence 1’, ‘sentence 2’ , ‘sentence 3’ ……. ‘Sentence n’]}

* + The tf\_idf function calculates top important words from all corpus files and returns 50 most common words.

**Function Descriptions:**

* **calc\_similarity():**

Calculates the similarity between two strings using the difflib function. Used to reduce redundancy in storing strings by comparing the similarity of URLs.

* **get\_urls(starter\_url):**

Takes a URL as input and returns a list of related URLs. Ignores irrelevant URLs based on user specified criteria (e.g., not containing 'wiki', 'donate', etc.).

* **visible(element):**

Determines if an HTML element is visible on a webpage. Used to filter out invisible elements during data scraping.

* **datascraper(my\_url, counter):**

Scrapes data from a given URL and saves it to a text file. Special cases are hardcoded – filmography table for corpus0 and Wikipedia page for corpus1. All the other corpuses are scraped with common code.

* **clean\_text(text\_string):**

Cleans the text in each file by removing references, non-alphanumeric characters, and extra whitespaces.

* **should\_append\_line(i, flag):**

Determines whether a line of text should be appended to the final text based on certain criteria (e.g., containing specific strings, not matching regex patterns).

* **filter\_data(i):**

Filters the data in a text file to remove irrelevant information. Uses should\_append\_line to decide which lines to keep.

* **main():**

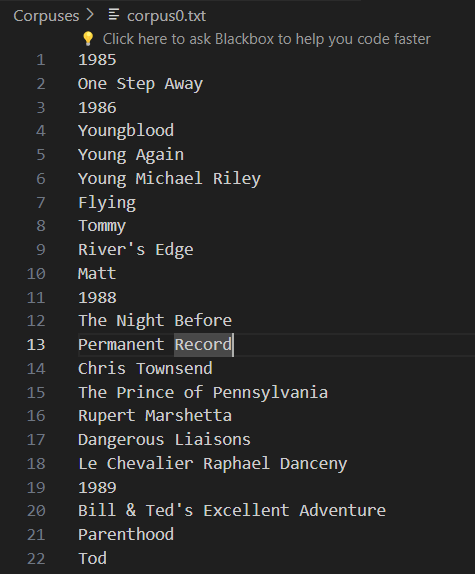
The main function that orchestrates the entire flow of the program. Collects URLs, scrapes data, cleans and filters the data, and saves the final URLs to a file.

**NLP Tools Used:**

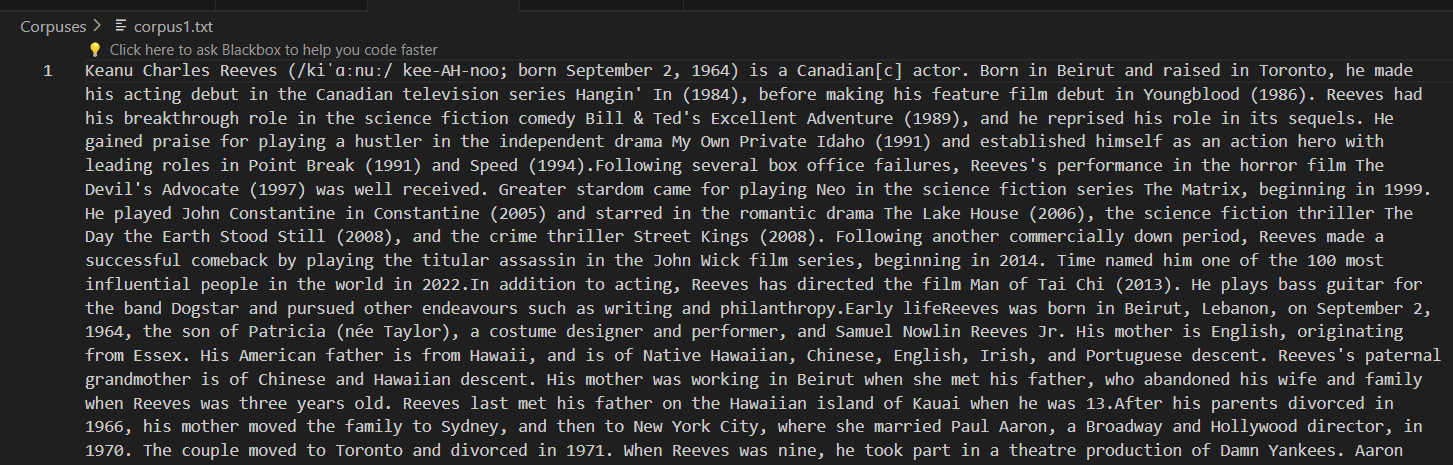
* **Regular Expressions (regex):** Used extensively for text cleaning and filtering, such as removing references, non-alphanumeric characters, and irrelevant lines of text.
* **Cosine Similarity:** Used to measure the similarity between two URLs to reduce redundancy in URL collection.
* **Information Extraction:** Utilized in functions like datascraper and filter\_data to extract relevant information from the scraped web pages, such as details from Keanu Reeves' filmography and Wikipedia page.
* **Beautiful Soup:** A Python library for pulling data out of HTML and XML files. It is used for parsing the HTML content of web pages to facilitate information extraction and data cleaning.
* **URLlibrary:** Library used to parse through websites and access data.
* **Pickle:** Used to store and retrieve pre-processed data for efficient access during runtime.

**Screenshots of Knowledge base:**

* corpus1.txt:

* corpus2.txt



* corpus3.txt

A screen shot of a computer screen

Description automatically generated

* movies.pickle

A screen shot of a computer

Description automatically generated

* tvshows.pickle

A screen shot of a computer

Description automatically generated

* Important words list:

A black background with white text

Description automatically generated

* List of files stored:

A screenshot of a computer

Description automatically generated