It is a web scraping and text search application. It has two major parts. The first part is web scraping, which uses Python libraries such as BeautifulSoup and requests to scrape publication data from a Coventry University website. The second part of the code is a search function that uses a search query and an inverted index to match queries to document IDs and then outputs the matching documents from the scraped publication data.

The code first imports all necessary libraries and sets up an empty list publication\_links to store the publication links to scrape. It then makes a GET request to the Coventry University website using the get() function from the requests library and saves the page content as a variable page\_source. The BeautifulSoup() function from the bs4 library is then used to parse the HTML content of page\_source. The script then scrapes the publication links from the parsed HTML using a for loop and the find\_all() function.

The script then sets up a new list details\_map and another for loop to scrape more publication links from subsequent pages of the Coventry University website. It appends these new links to publication\_links.

Next, the script sets up another empty list conference\_journals and a for loop that iterates over each publication link in publication\_links. It then makes a GET request to each publication link and saves the HTML content as html\_content. The script then parses this content using BeautifulSoup() and scrapes the conference journal links from the parsed HTML.

The script then sets up two new lists, conference\_journals1 and publications, and a new for loop that iterates over each conference journal link in conference\_journals. It then makes a GET request to each conference journal link and saves the HTML content as journal\_content. The script then parses this content using BeautifulSoup() and scrapes the journal title and authors from the parsed HTML.

After all the data is scraped, the script creates a Pandas DataFrame from the scraped data and saves it to a CSV file.

The second part of the code begins by importing more Python libraries such as NLTK, string, Flask, and collections. It then sets up a stemmer object from the PorterStemmer class, downloads the NLTK stopwords corpus and tokenizes the input query text.

The script then defines the i\_idx() function to create an inverted index from the scraped publication data. It iterates over each article title and preprocesses the text data by tokenizing, removing stopwords, and stemming the words. It then checks if the word is already in the inverted index and updates the index accordingly. The function then returns an ordered dictionary of the inverted index.

The script then defines the preprocess\_query() function to preprocess the user's search query. It tokenizes, removes stopwords, and stems the words.

Next, the script defines the search() function to match the preprocessed query terms to document IDs using the inverted index.

Finally, the script defines the search\_results1() function to retrieve the matching documents from the scraped publication data using the document IDs obtained from the search() function. It reads the scraped data from the CSV file into a Pandas DataFrame and filters the DataFrame based on the matching document IDs.