Opening

“Good day everyone. Today, I will present our URL Scraper application. This is a simple desktop program written in Python using Tkinter. The goal of this application is to take a website link, read the webpage, and extract useful information in an organized way.”

Program Overview

“This program allows the user to enter a website URL into the application. Once the user clicks the Scrape button or presses Enter, the program downloads the webpage and analyzes its content. It then extracts the page title, selected headings, email addresses found in the text, and a list of external links. All of this information is shown inside the application, and the user also has the option to save the results into a CSV file.”

How It Works

“Behind the interface, the program uses the requests library to connect to the website and download its content. It then uses BeautifulSoup to read and understand the HTML structure of the page. From there, the program finds the title tag, heading tags such as h1, h2, and h3, searches for email patterns using regular expressions, and collects external links that start with http. To keep the output clean, duplicate items are removed and limits are applied.”

Why This Program Is Useful

“This project is useful because it shows how different Python concepts work together. It combines a graphical interface, web requests, HTML parsing, regular expressions, and file exporting in one program. It also shows a complete workflow from user input, to data processing, and finally to displaying and saving results.”

Libraries and Setup

“Tkinter is already included in most Python installations, so the interface usually works without extra setup. For web scraping, the program uses two external libraries: requests and beautifulsoup4. If these libraries are not installed, the program detects this and shows a message telling the user what needs to be installed.”

Program Structure

“The application is written as a single-file Tkinter program. It has three main actions. The first is the scrape function, which downloads the webpage, extracts the data, and displays it. The second is the save function, which stores the scraped data into a CSV file. The third is the clear function, which resets the interface so the user can start again. The data flows from the input box, through the scraping process, and finally to the output area and the saved file.”

Scraping Process

“When scraping starts, the program first checks the URL entered by the user. If the link does not start with http or https, the program automatically adds https to avoid errors. The Scrape button is temporarily disabled to prevent repeated clicks. The program then sends the request using a safe User-Agent and a timeout. After receiving the response, it extracts the title, headings, email addresses, and links, then displays them clearly in the output area.”

Error Handling

“Web requests can fail for many reasons such as network issues or invalid URLs. The program handles these cases by catching errors and showing clear messages to the user. Even if an error happens, the interface is restored so the user can try again. This makes the program more user-friendly.”

Saving and Clearing Data

“Saving is only allowed after a successful scrape. When the Save button is clicked, the program creates a CSV file with a timestamp in the filename so previous data is not overwritten. The Clear button resets the input field, clears the output, disables saving again, and removes old data. This helps avoid mistakes.”

Demonstration

“For the demo, I can enter a public website such as python.org or wikipedia.org. After clicking Scrape, you can see the page title, headings, and link list appear. Then I can save the data to a CSV file and clear the screen to start again.”

Limitations and Improvements

“This program has some limitations. Some websites block scraping or require JavaScript, which this program does not handle. Email detection may not always be perfect, and the interface can pause briefly during slow requests. In the future, this can be improved by using background threads, better link handling, more export formats, and progress indicators.”

Closing

“To conclude, this project demonstrates a complete pipeline from graphical user input, to web data extraction, and finally to file export. It keeps the code simple while using real tools commonly used in practice. Thank you for listening.”