Web Scraper Tool

CODE :

```
import requests
from bs4 import BeautifulSoup
import threading
import tkinter as tk
from tkinter import scrolledtext, messagebox, filedialog
from tkinter import ttk  # Import ttk for Progressbar
import time  # Import time for sleep functionality


class WebScraperApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Advanced Web Scraper")
        self.root.geometry("1000x700")
        self.root.configure(bg="#f0f0f0")


        # Title Label
        self.title_label = tk.Label(root, text="Web Scraper", bg="#f0f0f0",
font=("Arial", 24, "bold"))
        self.title_label.pack(pady=10)


        # Frame for input and buttons
        self.input_frame = tk.Frame(root, bg="#f0f0f0")
```

```
        self.input_frame.pack(pady=20)

        self.url_label = tk.Label(self.input_frame, text="Enter URL:",
bg="#f0f0f0", font=("Arial", 12))
        self.url_label.pack(side=tk.LEFT, padx=5)

        self.url_entry = tk.Entry(self.input_frame, width=50,
font=("Arial", 12))
        self.url_entry.pack(side=tk.LEFT, padx=5)

        self.delay_label = tk.Label(self.input_frame, text="Delay
(seconds):", bg="#f0f0f0", font=("Arial", 12))
        self.delay_label.pack(side=tk.LEFT, padx=5)

        self.delay_entry = tk.Entry(self.input_frame, width=5,
font=("Arial", 12))
        self.delay_entry.pack(side=tk.LEFT, padx=5)
        self.delay_entry.insert(0, "1")  # Default delay of 1 second

        self.scrape_button = tk.Button(self.input_frame, text="Scrape
Data", command=self.start_scraping, bg="#4CAF50", fg="white",
font=("Arial", 12))
        self.scrape_button.pack(side=tk.LEFT, padx=5)
```

```
        self.save_button = tk.Button(self.input_frame, text="Save Results", command=self.save_results, state=tk.DISABLED, bg="#2196F3", fg="white", font=("Arial", 12))
        self.save_button.pack(side=tk.LEFT, padx=5)

        # Progress Bar
        self.progress = ttk.Progressbar(root, orient="horizontal", length=800, mode="determinate")
        self.progress.pack(pady=20)

        # Frame for output
        self.output_frame = tk.Frame(root, bg="#f0f0f0")
        self.output_frame.pack(pady=10)

        # Links Output
        self.links_label = tk.Label(self.output_frame, text="Found Links:", bg="#f0f0f0", font=("Arial", 14, "bold"))
        self.links_label.grid(row=0, column=0, padx=10, sticky='w')
        self.links_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)
        self.links_area.grid(row=1, column=0, padx=10, pady=5)

        # Images Output
        self.images_label = tk.Label(self.output_frame, text="Found Images:", bg="#f0f0f0", font=("Arial", 14, "bold"))
```

```
        self.images_label.grid(row=0, column=1, padx=10, sticky='w')
        self.images_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)
        self.images_area.grid(row=1, column=1, padx=10, pady=5)

        # Headings Output
        self.headings_label = tk.Label(self.output_frame, text="Found Headings:", bg="#f0f0f0", font=("Arial", 14, "bold"))
        self.headings_label.grid(row=2, column=0, padx=10, sticky='w')
        self.headings_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)
        self.headings_area.grid(row=3, column=0, padx=10, pady=5)

        # Meta Tags Output
        self.meta_tags_label = tk.Label(self.output_frame, text="Found Meta Tags:", bg="#f0f0f0", font=("Arial", 14, "bold"))
        self.meta_tags_label.grid(row=2, column=1, padx=10, sticky='w')
        self.meta_tags_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)
        self.meta_tags_area.grid(row=3, column=1, padx=10, pady=5)

        self.links = []  # Store scraped links
        self.images = []  # Store scraped images
        self.headings = []  # Store scraped headings
```

```
        self.meta_tags = []  # Store scraped meta tags

    def start_scraping(self):
        url = self.url_entry.get()
        if url:
            self.clear_output()  # Clear previous results
            threading.Thread(target=self.scrape_data, args=(url,)).start()
        else:
            messagebox.showwarning("Input Error", "Please enter a valid
URL.")

    def clear_output(self):
        self.links.clear()
        self.images.clear()
        self.headings.clear()
        self.meta_tags.clear()
        self.links_area.delete(1.0, tk.END)
        self.images_area.delete(1.0, tk.END)
        self.headings_area.delete(1.0, tk.END)
        self.meta_tags_area.delete(1.0, tk.END)
        self.progress['value'] = 0  # Reset progress bar

    def scrape_data(self, url):
```

```
        delay = float(self.delay_entry.get())  # Get the delay from the input field
        try:
            response = requests.get(url)
            response.raise_for_status()  # Raise an error for bad responses
            soup = BeautifulSoup(response.text, 'html.parser')

            total_items = len(soup.find_all('a', href=True)) + len(soup.find_all('img', src=True)) + \
                          sum(len(soup.find_all(f'h{i}')) for i in range(1, 7)) + \
                          len(soup.find_all('meta'))

            current_item = 0

            # Scrape links
            for link in soup.find_all('a', href=True):
                self.links.append(link['href'])
                current_item += 1
                self.update_progress(current_item, total_items)
                time.sleep(delay)  # Rate limiting

            # Scrape images
            for img in soup.find_all('img', src=True):
                self.images.append(img['src'])
```

```
                current_item += 1
                self.update_progress(current_item, total_items)
                time.sleep(delay)  # Rate limiting

            # Scrape headings
            for i in range(1, 7):  # h1 to h6
                for heading in soup.find_all(f'h{i}'):
                    self.headings.append(heading.get_text(strip=True))
                    current_item += 1
                    self.update_progress(current_item, total_items)
                    time.sleep(delay)  # Rate limiting

            # Scrape meta tags
            for meta in soup.find_all('meta'):
                if 'name' in meta.attrs:
                    self.meta_tags.append(f"{meta['name']}: 
{meta.get('content', '')}")
                    current_item += 1
                    self.update_progress(current_item, total_items)
                    time.sleep(delay)  # Rate limiting

            self.display_results()
            self.save_button.config(state=tk.NORMAL)  # Enable save 
button
```

```
        except requests.exceptions.RequestException as e:
            messagebox.showerror("Error", f"An error occurred: {e}")

    def update_progress(self, current, total):
        """Update the progress bar based on the current and total items."""
        progress_percentage = (current / total) * 100
        self.progress['value'] = progress_percentage
        self.root.update_idletasks()

    def display_results(self):
        # Display links with numbering
        if self.links:
            for i, link in enumerate(self.links, start=1):
                self.links_area.insert(tk.END, f"{i}. {link}\n")
        else:
            self.links_area.insert(tk.END, "No links found.")

        # Display images with numbering
        if self.images:
            for i, img in enumerate(self.images, start=1):
                self.images_area.insert(tk.END, f"{i}. {img}\n")
        else:
            self.images_area.insert(tk.END, "No images found.")
```

```
        # Display headings with numbering
        if self.headings:
            for i, heading in enumerate(self.headings, start=1):
                self.headings_area.insert(tk.END, f"{i}. {heading}\n")
        else:
            self.headings_area.insert(tk.END, "No headings found.")

        # Display meta tags with numbering
        if self.meta_tags:
            for i, meta in enumerate(self.meta_tags, start=1):
                self.meta_tags_area.insert(tk.END, f"{i}. {meta}\n")
        else:
            self.meta_tags_area.insert(tk.END, "No meta tags found.")

    def save_results(self):
        file_type = [('Text files', '*.txt'), ('CSV files', '*.csv')]
        file_path = filedialog.asksaveasfilename(defaultextension=".txt",
filetypes=file_type)

        if file_path:
            try:
                with open(file_path, 'w', newline='') as file:
                    file.write("Found Links:\n")
```

```
                for i, link in enumerate(self.links, start=1):
                    file.write(f"{i}. {link}\n")

                file.write("\nFound Images:\n")
                for i, img in enumerate(self.images, start=1):
                    file.write(f"{i}. {img}\n")

                file.write("\nFound Headings:\n")
                for i, heading in enumerate(self.headings, start=1):
                    file.write(f"{i}. {heading}\n")

                file.write("\nFound Meta Tags:\n")
                for i, meta in enumerate(self.meta_tags, start=1):
                    file.write(f"{i}. {meta}\n")

            messagebox.showinfo("Success", "Results saved
successfully.")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred while
saving: {e}")

if __name__ == "__main__":
    root = tk.Tk()
    app = WebScraperApp(root)
```

```
root.mainloop()
```

Code Breakdown :

1. Importing Required Libraries

Code :
```
import requests
    from bs4 import BeautifulSoup
    import threading
    import tkinter as tk
    from tkinter import scrolledtext, messagebox, filedialog
    from tkinter import ttk  # Import ttk for Progressbar
    import time  # Import time for sleep functionality
```

- **requests**: A library for making HTTP requests to fetch web pages.
- **BeautifulSoup**: A library for parsing HTML and XML documents, allowing easy extraction of data.
- **threading**: A module that allows the creation of threads, enabling concurrent execution of code.
- **tkinter**: A standard GUI toolkit in Python for creating graphical user interfaces.
- **scrolledtext**: A widget in tkinter that provides a text area with a scrollbar.

- **messagebox**: A module in tkinter for displaying message boxes.
- **filedialog**: A module in tkinter for opening file dialogs.
- **ttk**: A module in tkinter that provides themed widgets, including the Progressbar.
- **time**: A module that provides various time-related functions, including sleep.

2. Class Definition

Code : class WebScraperApp:

- Defines a class named **WebScraperApp** that encapsulates the entire web scraper application.

3. Initialization Method

Code : def __init__(self, root):

- The constructor method initializes the application and sets up the GUI components.

Code : self.root = root

```
self.root.title("Advanced Web Scraper")
self.root.geometry(“1000x700”)
self.root.configure(bg=”#f0f0f0”)
```

- **self.root**: Stores the main window reference.
- **title**: Sets the title of the window.

- **geometry**: Sets the size of the window to 1000x700 pixels.
- **configure**: Sets the background color of the window.

4. Title Label

Code : self.title_label = tk.Label(root, text="Web Scraper", bg="#f0f0f0", font=("Arial", 24, "bold"))

self.title_label.pack(pady=10)

- Creates a label widget displaying "Web Scraper" with a specified background color and font size, and adds it to the window with padding.

5. Input Frame

Code : self.input_frame = tk.Frame(root, bg="#f0f0f0")

self.input_frame.pack(pady=20)

- Creates a frame to hold input widgets and buttons, and adds it to the main window.

6. URL Input

Code : self.url_label = tk.Label(self.input_frame, text="Enter URL:", bg="#f0f0f0", font=("Arial", 12))

self.url_label.pack(side=tk.LEFT, padx=5)

self.url_entry = tk.Entry(self.input_frame, width=50, font=("Arial", 12))

self.url_entry.pack(side=tk.LEFT, padx=5)

- Creates a label and an entry field for the user to input the URL to scrape.

7. Delay Input

Code : self.delay_label = tk.Label(self.input_frame, text="Delay (seconds):", bg="#f0f0f0", font=("Arial", 12))

self.delay_label.pack(side=tk.LEFT, padx=5)

self.delay_entry = tk.Entry(self.input_frame, width=5, font=("Arial", 12))

self.delay_entry.pack(side=tk.LEFT, padx=5)

self.delay_entry.insert(0, "1")

- Creates a label and an entry field for the user to specify a delay (in seconds) between requests, with a default value of 1 second.

8. Scrape Button

Code : self.scrape_button = tk.Button(self.input_frame, text="Scrape Data", command=self.start_scraping, bg="#4CAF50", fg="white", font=("Arial", 12))

self.scrape_button.pack(side=tk.LEFT, padx=5)

- Creates a button that, when clicked, will start the scraping process by calling the **start_scraping** method.

9. Save Button

Code : self.save_button = tk.Button(self.input_frame, text="Save Results", command=self.save_results, state=tk.DISABLED, bg="#2196F3", fg="white", font=("Arial", 12))

self.save_button.pack(side=tk.LEFT, padx=5)

- Creates a button to save the scraped results, initially disabled until scraping is complete.

10. Progress Bar

Code : self.progress = ttk.Progressbar(root, orient="horizontal", length=800, mode="determinate")

self.progress.pack(pady=20)

- Creates a horizontal progress bar to visually indicate the scraping progress.

11. Output Frame

Code : self.output_frame = tk.Frame(root, bg="#f0f0f0")

self.output_frame.pack(pady=10)

- Creates a frame to hold output widgets for displaying the scraped data.

12. Links Output

Code : self.links_label = tk.Label(self.output_frame, text="Found Links:", bg="#f0f0f0", font=("Arial", 14, "bold"))

self.links_label.grid(row=0, column=0, padx=10, sticky='w')

self.links_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)

self.links_area.grid(row=1, column=0, padx=10, pady=5)

- Creates a label and a scrolled text area to display the found links.

13. Images Output

Code : self.images_label = tk.Label(self.output_frame, text="Found Images:", bg="#f0f0f0", font=("Arial", 14, "bold"))

self.images_label.grid(row=0, column=1, padx=10, sticky='w')

self.images_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)

self.images_area.grid(row=1, column=1, padx=10, pady=5)

- Creates a label and a scrolled text area to display the found images.

14. Headings Output

Code : self.headings_label = tk.Label(self.output_frame, text="Found Headings:", bg="#f0f0f0", font=("Arial", 14, "bold"))

self.headings_label.grid(row=2, column=0, padx=10, sticky='w')

self.headings_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)

self.headings_area.grid(row=3, column=0, padx=10, pady=5)

- Creates a label and a scrolled text area to display the found headings.

15. Meta Tags Output

Code : self.meta_tags_label = tk.Label(self.output_frame, text="Found Meta Tags:", bg="#f0f0f0", font=("Arial", 14, "bold"))

self.meta_tags_label.grid(row=2, column=1, padx=10, sticky='w')

self.meta_tags_area = scrolledtext.ScrolledText(self.output_frame, width=50, height=15, font=("Arial", 12), bg="#ffffff", wrap=tk.WORD)

self.meta_tags_area.grid(row=3, column=1, padx=10, pady=5)

- Creates a label and a scrolled text area to display the found meta tags.

16. Data Storage

Code : self.links = []

self.images = []

```
        self.headings = []
        self.meta_tags = []
```

- Initializes empty lists to store the scraped data.

17. Start Scraping Method

Code :
```
def start_scraping(self):
        url = self.url_entry.get()
        if url:
            self.clear_output()  # Clear previous results
            threading.Thread(target=self.scrape_data, args=(url,)).start()
        else:
            messagebox.showwarning("Input Error", "Please enter a valid URL.")
```

- Retrieves the URL from the entry field.
- If a valid URL is provided, it clears previous results and starts a new thread to run the **scrape_data** method.
- If no URL is provided, it shows a warning message.

18. Clear Output Method

Code :
```
def clear_output(self):
        self.links.clear()
        self.images.clear()
        self.headings.clear()
```

```
        self.meta_tags.clear()
        self.links_area.delete(1.0, tk.END)
        self.images_area.delete(1.0, tk.END)
       self.headings_area.delete(1.0, tk.END)
        self.meta_tags_area.delete(1.0, tk.END)
        self.progress['value'] = 0  # Reset progress bar
```

- Clears the lists that store scraped data.
- Clears the text areas displaying the results.
- Resets the progress bar to 0.

19. Scrape Data Method

Code :
```
def scrape_data(self, url):
        delay = float(self.delay_entry.get())
        try:
            response = requests.get(url)
            response.raise_for_status()  # Raise an error for bad responses
            soup = BeautifulSoup(response.text, 'html.parser')
```

- Retrieves the delay value from the input field.
- Makes an HTTP GET request to the specified URL.
- Raises an error if the response status is not successful (e.g., 404 or 500).

- Parses the HTML content of the page using BeautifulSoup.

20. Total Items Calculation

Code : total_items = len(soup.find_all('a', href=True)) + len(soup.find_all('img', src=True)) + \

sum(len(soup.find_all(f'h{i}')) for i in range(1, 7)) + \

len(soup.find_all('meta'))

- Calculates the total number of items to scrape (links, images, headings, and meta tags) for progress tracking.

21. Current Item Counter

Code : current_item = 0

- Initializes a counter to keep track of the number of items scraped.

22. Scraping Links

Code : for link in soup.find_all('a', href=True):

self.links.append(link['href'])

current_item += 1

self.update_progress(current_item, total_items)

time.sleep(delay) # Rate limiting

- Iterates through all anchor tags (**<a>**) with an **href** attribute.
- Appends each link to the **self.links** list.
- Increments the **current_item** counter.
- Calls **update_progress** to update the progress bar.
- Sleeps for the specified delay to avoid overwhelming the server.

23. Scraping Images

Code :
```
for img in soup.find_all('img', src=True):
        self.images.append(img['src'])
        current_item += 1
        self.update_progress(current_item, total_items)
        time.sleep(delay)  # Rate limiting
```

- Similar to the links, but iterates through all image tags (**<img>**) with a **src** attribute and appends the image sources to **self.images**.

24. Scraping Headings

Code :
```
for i in range(1, 7):  # h1 to h6
        for heading in soup.find_all(f'h{i}'):
            self.headings.append(heading.get_text(strip=True))
            current_item += 1
            self.update_progress(current_item, total_items)
```

time.sleep(delay)

- Iterates through heading tags (**<h1>** to **<h6>**).
- Appends the text content of each heading to **self.headings**.

25. Scraping Meta Tags

Code : for meta in soup.find_all('meta'):

if 'name' in meta.attrs:

self.meta_tags.append(f"{meta['name']}: {meta.get('content', '')}")

current_item += 1

self.update_progress(current_item, total_items)

time.sleep(delay)

- Iterates through all meta tags and appends their name and content to **self.meta_tags**.

26. Display Results and Enable Save Button

Code : self.display_results()

self.save_button.config(state=tk.NORMAL) # Enable save button

- Calls the **display_results** method to show the scraped data in the GUI.
- Enables the save button to allow the user to save the results.

27. Exception Handling

Code : except requests.exceptions.RequestException as e:

```
    messagebox.showerror("Error", f"An error occurred: {e}")
```

- Catches any exceptions that occur during the HTTP request and displays an error message.

28. Update Progress Method

Code : def update_progress(self, current, total):

```
        progress_percentage = (current / total) * 100
        self.progress['value'] = progress_percentage
        self.root.update_idletasks()  # Update the GUI
```

- Calculates the percentage of progress based on the current and total items.
- Updates the progress bar value.
- Calls **update_idletasks** to refresh the GUI.

29. Display Results Method

Code : def display_results(self):

```
        if self.links:
            for i, link in enumerate(self.links, start=1):
                self.links_area.insert(tk.END, f"{i}. {link}\n")
        else:
            self.links_area.insert(tk.END, "No links found.")
```

- Displays the scraped links in the corresponding text area, numbering them.
- If no links are found, it displays a message indicating that.

30. Display Images, Headings, and Meta Tags

Code : if self.images:

```
        for i, img in enumerate(self.images, start=1):
            self.images_area.insert(tk.END, f"{i}. {img}\n")
    else:
        self.images_area.insert(tk.END, "No images found.")

    if self.headings:
        for i, heading in enumerate(self.headings, start=1):
            self.headings_area.insert(tk.END, f"{i}. {heading}\n")
    else:
        self.headings_area.insert(tk.END, "No headings found.")

    if self.meta_tags:
        for i, meta in enumerate(self.meta_tags, start=1):
            self.meta_tags_area.insert(tk.END, f"{i}. {meta}\n")
    else:
        self.meta_tags_area.insert(tk.END, "No meta tags found.")
```

- Similar to links, it displays images, headings, and meta tags in their respective text areas, numbering them and providing messages if none are found.

31. Save Results Method

Code : def save_results(self):

```
        file_type = [('Text files', '*.txt'), ('CSV files', '*.csv')]
        file_path =
filedialog.asksaveasfilename(defaultextension=".txt",
filetypes=file_type)
```

- Opens a file dialog to allow the user to choose a location and filename to save the results, with options for text or CSV files.

32. Writing to File

Code :

```
if file_path:
    try:
        with open(file_path, 'w', newline='') as file:
            file.write("Found Links:\n")
            for i, link in enumerate(self.links, start=1):
                file.write(f"{i}. {link}\n")

            file.write("\nFound Images:\n")
            for i, img in enumerate(self.images, start=1):
```

```
                    file.write(f"{i}. {img}\n")

                file.write("\nFound Headings:\n")
                for i, heading in enumerate(self.headings,
start=1):
                    file.write(f"{i}. {heading}\n")

                file.write("\nFound Meta Tags:\n")
                for i, meta in enumerate(self.meta_tags,
start=1):
                    file.write(f"{i}. {meta}\n")
```

- If a valid file path is provided, it opens the file in write mode and writes the scraped results (links, images, headings, and meta tags) to the file.

33. Success Message

Code : messagebox.showinfo("Success", "Results saved successfully.")

except Exception as e:

messagebox.showerror("Error", f"An error occurred while saving: {e}")

- Displays a success message if the results are saved successfully.
- Catches any exceptions during the file writing process and shows an error message.

34. Main Execution Block

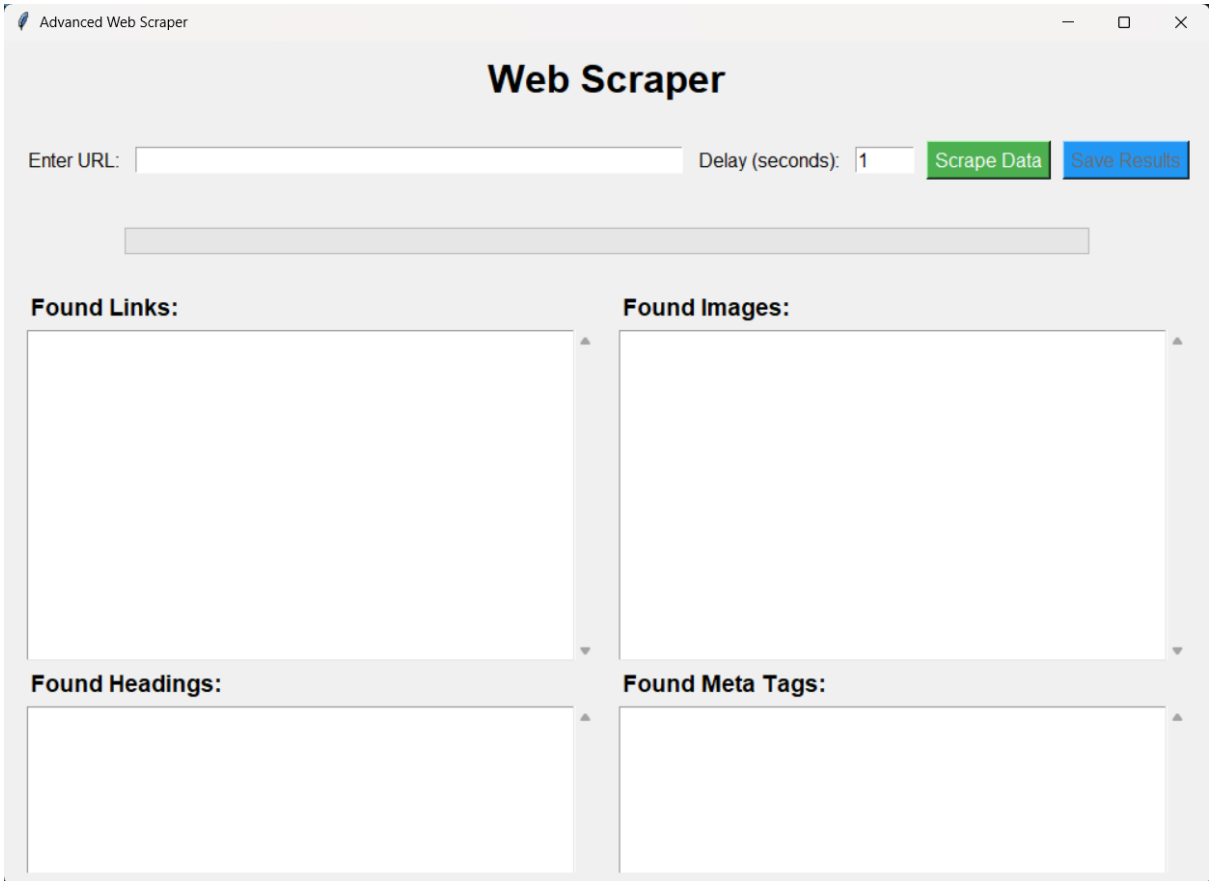
Code : if __name__ == "__main__":

```
root = tk.Tk()
app = WebScraperApp(root)
root.mainloop()
```

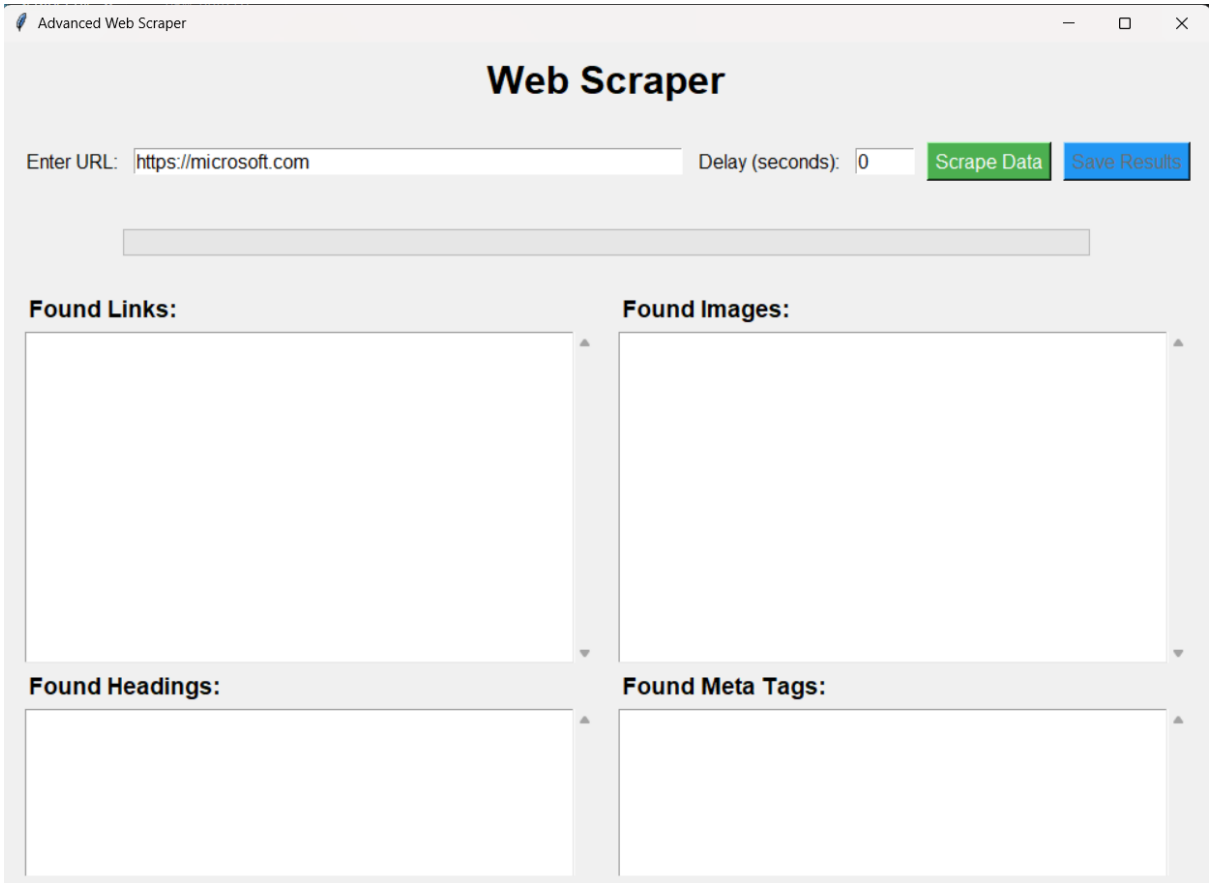
- Checks if the script is being run directly (not imported).
- Creates the main Tkinter window.
- Instantiates the **WebScraperApp** class, which sets up the GUI.
- Enters the Tkinter main loop, waiting for user interaction.

Screenshots :

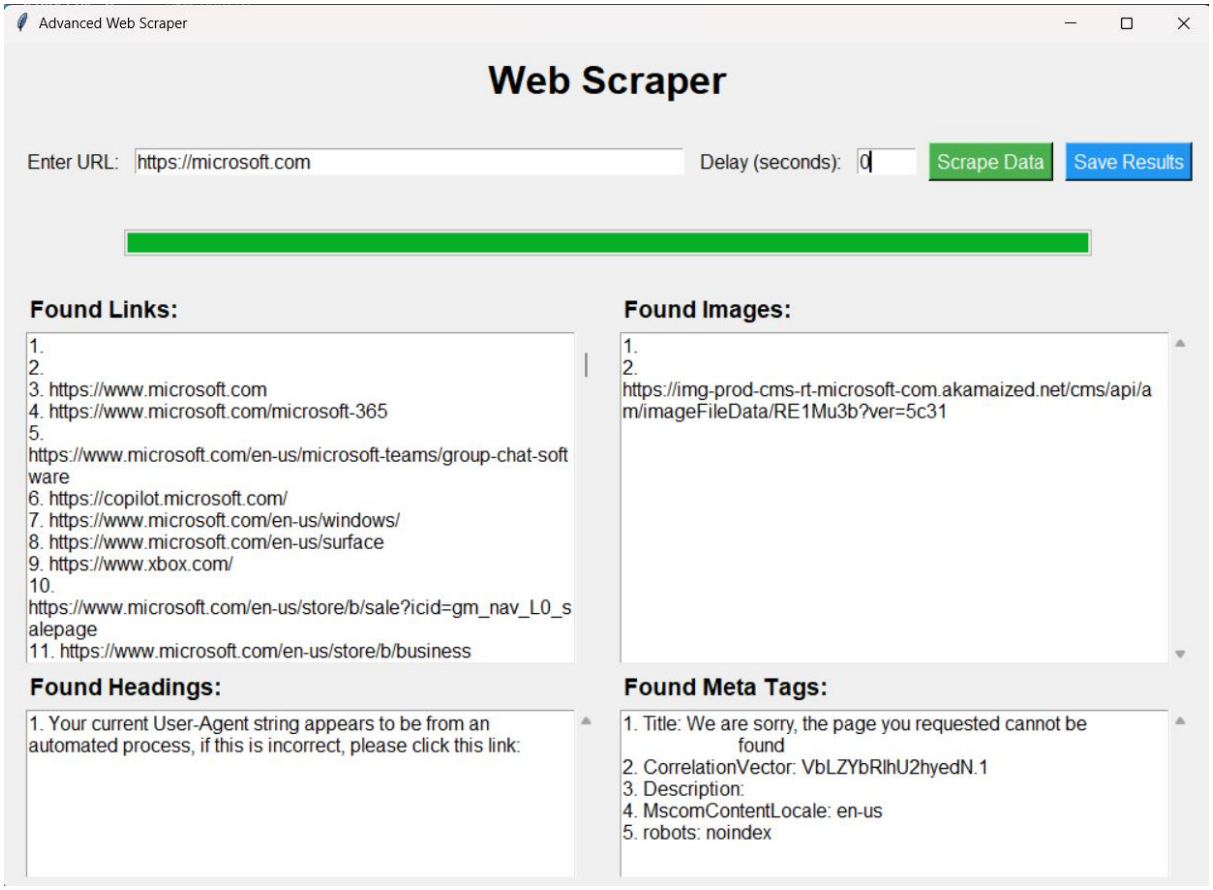
i. This image shows gui of web scraper tool



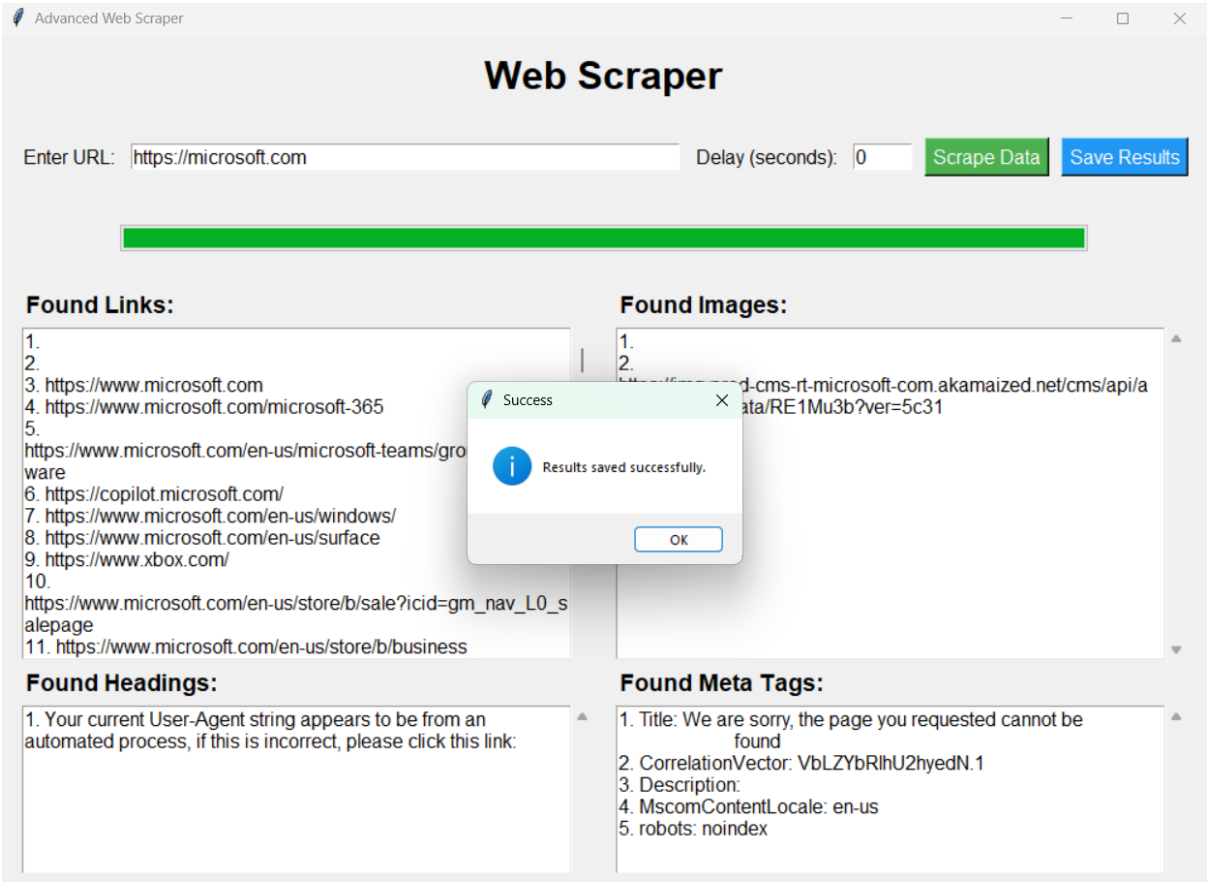
ii. In this image Microsoft.com is set as URL and delay is set 0 second



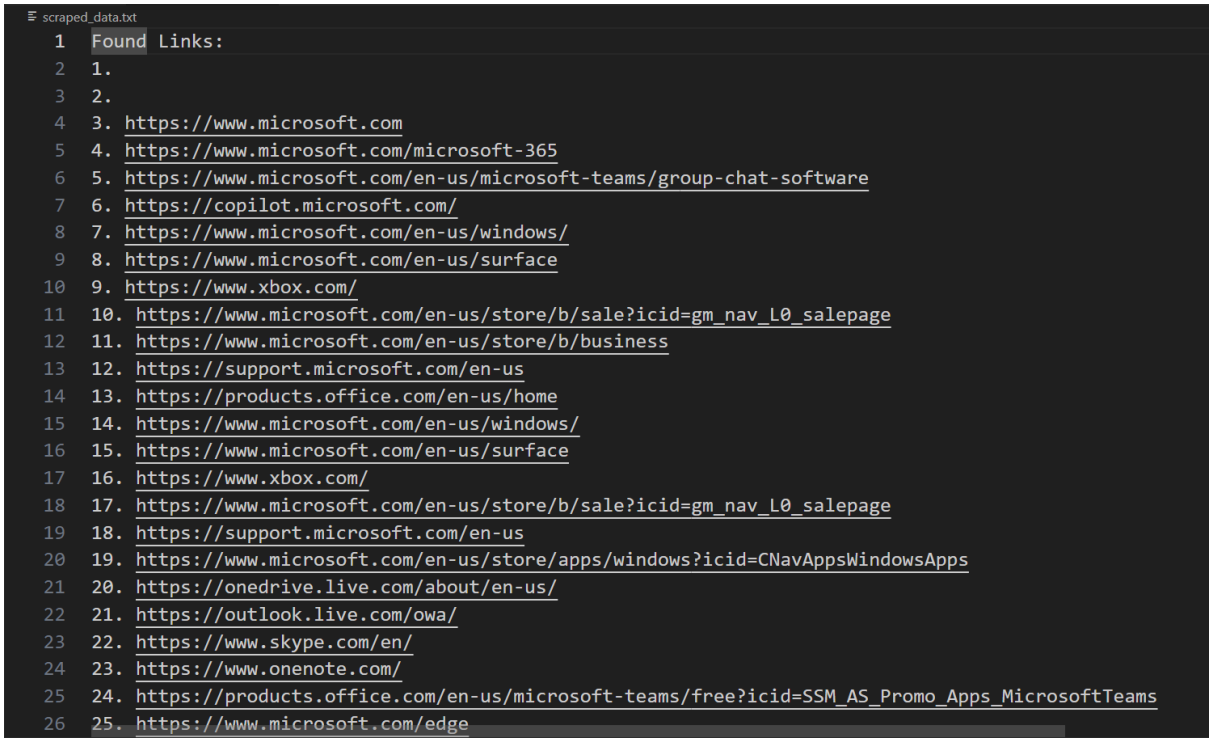
iii. This image shows output, this tool has found links, images, headings and meta tags



iv. This tool can save this scraped data in txt or csv format file



v. This image shows saved txt file with scraped data



scraped_data.txt

```
Found Links:
1.
2.
3. https://www.microsoft.com
4. https://www.microsoft.com/microsoft-365
5. https://www.microsoft.com/en-us/microsoft-teams/group-chat-software
6. https://copilot.microsoft.com/
7. https://www.microsoft.com/en-us/windows/
8. https://www.microsoft.com/en-us/surface
9. https://www.xbox.com/
10. https://www.microsoft.com/en-us/store/b/sale?icid=gm_nav_L0_salepage
11. https://www.microsoft.com/en-us/store/b/business
12. https://support.microsoft.com/en-us
13. https://products.office.com/en-us/home
14. https://www.microsoft.com/en-us/windows/
15. https://www.microsoft.com/en-us/surface
16. https://www.xbox.com/
17. https://www.microsoft.com/en-us/store/b/sale?icid=gm_nav_L0_salepage
18. https://support.microsoft.com/en-us
19. https://www.microsoft.com/en-us/store/apps/windows?icid=CNavAppsWindowsApps
20. https://onedrive.live.com/about/en-us/
21. https://outlook.live.com/owa/
22. https://www.skype.com/en/
23. https://www.onenote.com/
24. https://products.office.com/en-us/microsoft-teams/free?icid=SSM_AS_Promo_Apps_MicrosoftTeams
25. https://www.microsoft.com/edge
```