Python Project 12: Web Scraper

1. Project Overview

Build a simple **Web Scraper** that accepts a webpage URL, downloads its HTML, extracts the page title and links, and displays the results in a desktop interface.

Python's `urllib.request` can fetch URLs, while `html.parser` provides a built-in HTML parser.

For this project, the scraper uses Python's standard library, so no external package is required.

2. Features

- Enter a webpage URL
- Fetch webpage HTML
- Extract page title
- Extract links
- Display link text and URL
- Limit displayed results
- Error handling
- Scrollable results
- Simple Tkinter interface
- No external packages

3. Technologies

- Python 3
- Tkinter
- `urllib.request`
- `html.parser`
- Object Oriented Programming

4. Folder Structure

```
web_scraper/
□□□ main.py
□□□ ui.py
□□□ scraper.py
□□□ README.md
```

5. How the Project Works

```
User enters URL
      □
Web Scraper
      □
Send HTTP request
      □
Receive HTML
      □
HTMLParser processes page
      □
Extract title + links
      □
Tkinter displays results
```

The scraper sends a GET request and reads the returned HTML. `urllib.request` also supports adding request headers such as a User Agent.

6. Complete Backend Code

scraper.py

```
from html.parser import HTMLParser
from urllib.error import HTTPError, URLError
from urllib.parse import urljoin
```

```
from urllib.request import Request, urlopen


class PageParser(HTMLParser):

    def __init__(self):
        super().__init__()

        self.title = ""
        self.links = []

        self.inside_title = False

    def handle_starttag(self, tag, attrs):

        if tag.lower() == "title":
            self.inside_title = True

        if tag.lower() == "a":

            attributes = dict(attrs)
            href = attributes.get("href")

            if href:
                self.links.append({
                    "href": href,
                    "text": ""
                })

    def handle_endtag(self, tag):

        if tag.lower() == "title":
            self.inside_title = False
```

```
    def handle_data(self, data):

        if self.inside_title:
            self.title += data.strip()

        elif self.links and data.strip():
            self.links[-1]["text"] += data.strip()


class WebScraper:

    def fetch_page(self, url):

        if not url.startswith(("http://", "https://")):
            url = "https://" + url

        request = Request(
            url,
            headers={
                "User-Agent": (
                    "Mozilla/5.0 "
                    "(Windows NT 10.0; Win64; x64) "
                    "AppleWebKit/537.36 "
                    "Chrome/120 Safari/537.36"
                )
            }
        )

        try:
            with urlopen(request, timeout=10) as response:
                html = response.read().decode(
                    "utf-8",
                    errors="replace"
                )
```

```
            final_url = response.geturl()

        except HTTPError as error:
            raise Exception(
                f"HTTP error: {error.code}"
            )

        except URLError as error:
            raise Exception(
                f"Could not access webpage: {error.reason}"
            )

        except TimeoutError:
            raise Exception(
                "Request timed out."
            )

        parser = PageParser()
        parser.feed(html)

        for link in parser.links:
            link["href"] = urljoin(
                final_url,
                link["href"]
            )

        return {
            "title": parser.title or "No title found",
            "links": parser.links
        }
```

7. Complete Frontend Code

ui.py

```
import tkinter as tk
from tkinter import messagebox, ttk

from scraper import WebScraper


class WebScraperUI:

    def __init__(self, root):

        self.root = root
        self.root.title("Web Scraper")
        self.root.geometry("850x600")
        self.root.configure(bg="#0f172a")

        self.scraper = WebScraper()

        self.url_var = tk.StringVar()

        self.create_ui()

    def create_ui(self):

        title = tk.Label(
            self.root,
            text="Web Scraper",
            font=("Arial", 26, "bold"),
            bg="#0f172a",
            fg="white"
        )
        title.pack(pady=(30, 5))

        subtitle = tk.Label(
            self.root,
```

```
            text="Extract webpage title and links",
            font=("Arial", 11),
            bg="#0f172a",
            fg="#94a3b8"
        )
        subtitle.pack()

        search_frame = tk.Frame(
            self.root,
            bg="#0f172a"
        )
        search_frame.pack(
            fill="x",
            padx=50,
            pady=25
        )

        url_entry = tk.Entry(
            search_frame,
            textvariable=self.url_var,
            font=("Arial", 12),
            bg="#1e293b",
            fg="white",
            insertbackground="white",
            relief="flat"
        )
        url_entry.pack(
            side="left",
            fill="x",
            expand=True,
            ipady=10,
            padx=(0, 10)
        )
```

```
        url_entry.bind(
            "<Return>",
            lambda event: self.scrape_page()
        )

        scrape_button = tk.Button(
            search_frame,
            text="Scrape",
            command=self.scrape_page,
            font=("Arial", 11, "bold"),
            bg="#2563eb",
            fg="white",
            relief="flat",
            padx=25,
            pady=9
        )
        scrape_button.pack(side="right")

        self.title_label = tk.Label(
            self.root,
            text="Page Title: --",
            font=("Arial", 15, "bold"),
            bg="#0f172a",
            fg="white",
            anchor="w"
        )
        self.title_label.pack(
            fill="x",
            padx=50,
            pady=(0, 15)
        )

        table_frame = tk.Frame(
            self.root,
```

```
            bg="#1e293b"
        )
        table_frame.pack(
            fill="both",
            expand=True,
            padx=50
        )

        columns = ("text", "url")

        self.tree = ttk.Treeview(
            table_frame,
            columns=columns,
            show="headings"
        )

        self.tree.heading(
            "text",
            text="Link Text"
        )

        self.tree.heading(
            "url",
            text="URL"
        )

        self.tree.column(
            "text",
            width=250
        )

        self.tree.column(
            "url",
            width=500
```

```
        )

        scrollbar = ttk.Scrollbar(
            table_frame,
            orient="vertical",
            command=self.tree.yview
        )

        self.tree.configure(
            yscrollcommand=scrollbar.set
        )

        self.tree.pack(
            side="left",
            fill="both",
            expand=True
        )

        scrollbar.pack(
            side="right",
            fill="y"
        )

        self.status_label = tk.Label(
            self.root,
            text="Enter a URL to begin.",
            font=("Arial", 10),
            bg="#0f172a",
            fg="#94a3b8"
        )
        self.status_label.pack(pady=15)

    def scrape_page(self):
```

```
        url = self.url_var.get().strip()

        if not url:
            messagebox.showwarning(
                "Missing URL",
                "Please enter a webpage URL."
            )
            return

        self.status_label.config(
            text="Fetching webpage..."
        )

        self.root.update_idletasks()

        try:
            result = self.scraper.fetch_page(url)

            self.title_label.config(
                text=f"Page Title: {result['title']}"
            )

            for item in self.tree.get_children():
                self.tree.delete(item)

            for link in result["links"][:100]:

                text = link["text"] or "(no text)"

                self.tree.insert(
                    "",
                    "end",
                    values=(
                        text,
```

```
                    link["href"]
                )
            )

            self.status_label.config(
                text=(
                    f"Found {len(result['links'])} "
                    f"link(s). Showing up to 100."
                )
            )

        except Exception as error:

            self.status_label.config(
                text="Scraping failed."
            )

            messagebox.showerror(
                "Scraping Error",
                str(error)
            )
```

8. Main Code

main.py

```
import tkinter as tk

from ui import WebScraperUI


def main():

    root = tk.Tk()
```

```
    WebScraperUI(root)

    root.mainloop()


if __name__ == "__main__":
    main()
```

9. README

README.md

```
# Web Scraper

A simple Python desktop web scraper.

## Features

- Enter webpage URL
- Extract page title
- Extract links
- Display link text and URLs
- Error handling

## Requirements

Python 3

No external packages are required.

## Run

python main.py
```

10. How Frontend + Backend Connect

ui.py

WebScraper.fetch_page()

urllib.request

Download HTML

PageParser

Extract title + links

ui.py

Display results

The PageParser class extends Python's HTMLParser and reacts to HTML start tags, end tags, and text data.

11. How to Run

Open the project directory:

```
cd web_scraper
```

Run:

```
python main.py
```

Enter a URL such as:

```
https://www.python.org/
```

Then click **Scrape**.

12. Basic Testing

Test with:

`https://www.python.org/`
`https://example.com/`

Also test:

- Empty URL
- Invalid URL
- Non existent webpage
- Website that blocks automated requests
- Website containing relative links

The scraper converts relative links into absolute URLs using `urljoin()`.

13. Important Scraping Rules

A web scraper should only collect information that the website permits you to access. Check the site's terms, access restrictions, and `robots.txt` before scraping at scale. Python also provides `urllib.robotparser` for working with `robots.txt`.

This project is intentionally a small educational scraper and does not attempt to bypass authentication, rate limits, CAPTCHAs, or anti bot systems.

14. Small Improvements

- Export scraped data to CSV
- Add keyword filtering
- Extract images
- Extract headings
- Add pagination

- Add multiple URL support
- Add request delay
- Add robots.txt checking
- Save scraping history
- Add asynchronous scraping for multiple pages

Visit haas.dev for more resources and guides.