Automate a Simple Task (e.g., Web Scraping) Using Python Scripts

Table of Contents

Introduction

Automation of repetitive tasks using Python can significantly improve efficiency. One common task to automate is web scraping, where a script extracts data from websites for various purposes, like gathering information or collecting structured data from online sources.

Problem Statement

Learn to automate web scraping using Python scripts to extract data from websites. In this example, we'll scrape data from a webpage using the `requests` and `BeautifulSoup` libraries.

Prerequisites

Software Requirement

- **Python 3.13.0**
  Download Python

- **Code Editor**
  A text editor or IDE like **Visual Studio Code (VS Code)** is recommended.
  Download VS Code

- **Command Line/Terminal**: For running Python scripts.

Hardware Requirement

- **Processor**: Minimum dual-core processor.
- **RAM**: 4GB or more.

- **Storage**: At least 1GB free space for Python and your project files.

Libraries Required

Before proceeding, you need to install some Python libraries that will help with web scraping:

- **requests**: For sending HTTP requests to fetch the webpage content.
- **beautifulsoup4**: For parsing the HTML content and extracting specific data.

Run the following commands in the terminal to install these libraries:

```
pip install requests
pip install beautifulsoup4
```

```
PS C:\Users\Administrator\Desktop\python> pip install requests
Collecting requests
  Using cached requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting charset-normalizer<4,>=2 (from requests)
  Using cached charset_normalizer-3.4.0-cp313-cp313-win_amd64.whl.metadata (34 kB)
Collecting idna<4,>=2.5 (from requests)
  Using cached idna-3.10-py3-none-any.whl.metadata (10 kB)
Collecting urllib3<3,>=1.21.1 (from requests)
  Using cached urllib3-2.2.3-py3-none-any.whl.metadata (6.5 kB)
Collecting certifi>=2017.4.17 (from requests)
  Using cached certifi-2024.8.30-py3-none-any.whl.metadata (2.2 kB)
Using cached requests-2.32.3-py3-none-any.whl (64 kB)
Using cached certifi-2024.8.30-py3-none-any.whl (167 kB)
Using cached charset_normalizer-3.4.0-cp313-cp313-win_amd64.whl (102 kB)
Using cached idna-3.10-py3-none-any.whl (70 kB)
Using cached urllib3-2.2.3-py3-none-any.whl (126 kB)
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests
Successfully installed certifi-2024.8.30 charset-normalizer-3.4.0 idna-3.10 requests-2.32.3 urllib3-2.2.3
```

```
PS C:\Users\Administrator\Desktop\python> pip install beautifulsoup4
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.12.3-py3-none-any.whl.metadata (3.8 kB)
Collecting soupsieve>1.2 (from beautifulsoup4)
  Downloading soupsieve-2.6-py3-none-any.whl.metadata (4.6 kB)
Downloading beautifulsoup4-4.12.3-py3-none-any.whl (147 kB)
Downloading soupsieve-2.6-py3-none-any.whl (36 kB)
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.12.3 soupsieve-2.6
```

Implementation Steps

Step-1: Set Up Your Python Environment

1. **Create a new Python file**: Start by creating a Python script named `web_scraper.py` in your project directory.
2. **Install required libraries**: As mentioned in the prerequisites, install the `requests` and `beautifulsoup4` libraries.

Step-2: Write a Python Script for Web Scraping

Add the following code to `web_scraper.py` to scrape and extract data from a webpage:

```
import requests
from bs4 import BeautifulSoup

def scrape_webpage(url):
    """Scrape the webpage and extract data."""
    try:
        # Send an HTTP request to the URL
        response = requests.get(url)
        # Check if the request was successful (status code 200)
        if response.status_code == 200:
            # Parse the webpage content
            soup = BeautifulSoup(response.content, 'html.parser')
            # Extract and return the webpage's title as an example
            title = soup.title.string
            return f"Webpage Title: {title}"
        else:
            return f"Failed to retrieve data. HTTP Status code:
{response.status_code}"
    except Exception as e:
        return f"An error occurred: {e}"


url = "https://www.geeksforgeeks.org/extract-all-the-urls-from-the-webpage-using-
python/?ref=lbp"
print(scrape_webpage(url))
```

- **requests.get(url)**: Sends an HTTP request to fetch the content of the webpage.
- **BeautifulSoup**: Parses the HTML content of the page and allows easy access to the data. In this case, we're extracting the title tag content as an example.
- **Error handling**: We use a try-except block to catch and handle potential errors such as connection failures.

Step-3: Run and Test the Script

To run the Python script, open your terminal and navigate to the directory where `web_scraper.py` is saved. Then, execute the script by running:

```
python web_scraper.py
```

Output:

```
PS C:\Users\Administrator\Desktop\python> python web_scraper.py
Webpage Title: Extract all the URLs from the webpage Using Python - GeeksforGeeks
```

You can modify the script to extract other elements from the webpage, such as headings, paragraphs, or links.

References

- BeautifulSoup Documentation
- Requests Library Documentation
