**Recursively Scraping Files from Websites with Python**

Web scraping is a powerful tool for automating data collection from websites. This blog explores how to build a Python script that recursively scrapes a website to download readable files like PDFs and text files. We will use the aiohttp library for asynchronous requests, BeautifulSoup for HTML parsing, and urllib.parse for handling URLs.

**Key Features of the Script**

1. **Extract Links:** Extract all hyperlinks (<a> tags) from a webpage.
2. **Identify Readable Links:** Filter links that point to PDF or text files.
3. **Download Files:** Download these files asynchronously and save them to a local folder.
4. **Recursive Scraping:** Crawl through the website recursively, following internal links.

**Prerequisites**

Before diving into the code, ensure you have Python 3.7+ installed on your system. Install the required libraries by running:

pip install aiohttp beautifulsoup4

**The Code**

**Importing Required Libraries**

import os

import aiohttp

import asyncio

from bs4 import BeautifulSoup

from urllib.parse import urljoin

These libraries provide the necessary functionality for handling HTTP requests, parsing HTML, and manipulating URLs.

**Function to Download Files**

async def download\_file(session, url, folder):

"""

Download a file from a given URL and save it to a folder.

"""

try:

async with session.get(url, ssl=False) as response:

response.raise\_for\_status()

file\_name = os.path.basename(url)

file\_path = os.path.join(folder, file\_name)

with open(file\_path, 'wb') as file:

async for chunk in response.content.iter\_chunked(8192):

file.write(chunk)

print(f"Downloaded: {file\_name}")

except Exception as e:

print(f"Failed to download {url}: {e}")

This function downloads files asynchronously. It handles exceptions to ensure the script doesn't crash if a file fails to download.

**Extracting Links**

**Extract All Links**

async def extract\_links(session, url):

"""

Extract all links from a webpage.

"""

try:

async with session.get(url, ssl=False) as response:

response.raise\_for\_status()

html = await response.text()

soup = BeautifulSoup(html, 'html.parser')

links = soup.find\_all('a', href=True)

full\_links = [urljoin(url, link['href']) for link in links]

return full\_links

except Exception as e:

print(f"Error extracting links from {url}: {e}")

return []

This function extracts all <a> tags from a webpage and converts relative URLs to absolute URLs using urljoin.

**Recursive Scraping**

async def scrape\_papers(session, url, folder, visited=None):

"""

Recursively scrape for paper links and download readable files.

"""

if visited is None:

visited = set()

if url in visited:

return

visited.add(url)

print(f"Scraping: {url}")

# Extract all links on the page

links = await extract\_links(session, url)

for link in links:

if link.endswith('.pdf') or link.endswith('.txt'): # Check if the link is for a readable file

await download\_file(session, link, folder)

else:

# Recursively explore other links

await scrape\_papers(session, link, folder, visited)

This function crawls through a website recursively, ensuring each URL is visited only once. It identifies and downloads files like PDFs and text files, then follows all other links to continue the scraping process.

**Main Function**

async def main():

# Starting URL for the website

base\_url = "https://papers.nips.cc/" # Replace with the target website

# Folder to save downloaded files

download\_folder = "downloaded\_papers"

os.makedirs(download\_folder, exist\_ok=True)

async with aiohttp.ClientSession(connector=aiohttp.TCPConnector(ssl=False)) as session:

await scrape\_papers(session, base\_url, download\_folder)

# Check if running in Jupyter or another environment

try:

# For Jupyter or environments with a running event loop

loop = asyncio.get\_running\_loop()

task = loop.create\_task(main())

loop.run\_until\_complete(task)

except RuntimeError:

# For standalone scripts

asyncio.run(main())

The main function initializes the download folder and starts the scraping process by calling scrape\_papers.

**How It Works**

1. **Start with a URL:** The script begins with the root URL of the website.
2. **Download Files:** It identifies and downloads all readable files (PDFs, text files) on the current page.
3. **Follow Links:** It extracts all links and recursively visits each one.
4. **Avoid Revisiting:** The visited set ensures that no URL is processed more than once.

**Example Output**

Running the script will create a downloaded\_papers folder and display logs like:

Scraping: https://papers.nips.cc/

Downloaded: paper1.pdf

Downloaded: document.txt

Scraping: https://papers.nips.cc/conference.html

...

**Challenges and Considerations**

1. **SSL Verification:** Disabled in this script for simplicity but should be enabled for secure websites.
2. **Large Websites:** For large websites, consider rate-limiting requests or using a depth limit for recursion.
3. **Error Handling:** Ensure robust handling for broken links, timeouts, and inaccessible pages.
4. **Legal and Ethical Issues:** Always check the website's robots.txt file and terms of service before scraping.

**Conclusion**

This Python script demonstrates how to automate the task of downloading readable files from a website. With aiohttp, the script efficiently handles multiple requests simultaneously, making it scalable for large websites. Modify the script as needed to suit your specific use case!