Python Development Intern's

"WE SPEAK DATA"

@interncareer

By Rashini Hettiarachchi

1: Python Script for Web Scraping and Automation

Introduction:

The primary objective of this project is to leverage online scraping techniques to extract valuable data from a chosen website, hence simplifying the process of creating and managing an updated dataset. eBay is a well-known website with a wealth of freely accessible content.

It makes it an excellent resource for relevant information.Our objective is to systematically collect certain data points using Python scripts powered by packages such as Beautiful Soup and Requests.This project was driven by the need for quick and reliable information retrieval, which forms the foundation for data analysis, market insights, and other pertinent applications.The automation part of the project ensures that the dataset is automatically updated by executing a script on a predetermined schedule (daily, weekly, etc.). ensures that the dataset is current and aligned with the changing content of the website. By describing the script's execution methods, dependencies, and other details, this explanation emphasizes the significance of the and make an effort. maintaining a sizable and up-to-date dataset from **the BBC website.**

Website Selection:

I have chosen **BBC website** for web scraping and headline extraction because of several reasons:

- **Public RSS Feeds:** Some news websites, including the BBC, offer public RSS feeds that make it easier to access and extract headlines without the need for extensive web scraping. Check if the BBC provides such feeds as an alternative approach.
- **Availability of Headline Data:** BBC's website often prominently displays headlines, making it convenient to locate and scrape this information compared to some other news websites where headlines might be nested deeper or displayed differently.
- **Personal Preference or Familiarity:** The choice of the BBC could also stem from personal preference, familiarity, or trust in the news organization, prompting the selection of this specific website for scraping news headlines.
- **Consistent Structure:** Well-established news websites often maintain a consistent structure for their articles and headlines. If the BBC website follows a standard format, it may make the web scraping process more manageable and predictable.
- **Structured HTML:** Websites differ in their HTML structure, and the BBC likely maintains a structured layout for headlines, making it relatively easier to extract specific information like headlines using BeautifulSoup or similar parsing libraries.

Web Scraping:

One method for obtaining data from webpages is web scraping. It entails retrieving webpages, interpreting their HTML or XML content, and then extracting the needed data for additional usage. This snippet of code uses web scraping to pull headlines from the BBC News website.

Here's a breakdown of how web scraping is used:

• **Fetching Web Content:** The requests library is utilized to send an HTTP GET request to the BBC News website in the **scrape_bbc_headlines** function. This retrieves the HTML content of the webpage. (https://www.bbc.com/news)

• **Parsing HTML Content**: The **BeautifulSoup** library is employed to parse the retrieved HTML content **(response.text).** This creates a parse tree that allows the program to navigate and search through the HTML structure to locate specific elements.

• **Machine Learning and Training Data:** Web scraping is often used to collect data for training machine learning models, such as image recognition, sentiment analysis, and natural language processing.

• **Extracting Headline Text:** Once the headline-containing elements are located, the code extracts the text from these elements using the **get_text()** method. The extracted text is stored in a list **(headline_texts)** for further processing.

- **Locating Headline Elements:** Finding Headline Elements: The code locates the HTML elements that hold the headlines by utilizing BeautifulSoup's features. Here, it looks for **<h3>** tags that have the

gs-c-promo-heading__title class. The headlines from the BBC News page are probably included in these components.

- **Language Translation and Sentiment Analysis:**Language service providers use web scraping to collect multilingual content for translation, and businesses analyze sentiment on social media for customer feedback.

- **Return and Display:** The extracted headlines are then returned from the function. If headlines are found, the code prints them in the console, showing each headline with an index.

- **Weather Data Retrieval:** Meteorological services use web scraping to collect weather data from multiple sources, helping improve accuracy in forecasting.

- **Real Estate Data Collection:** Real estate professionals use web scraping to gather information on property listings, market trends, and property prices from various real estate websites.

Python code scripts:

```
import requests

from bs4 import BeautifulSoup
from apscheduler.schedulers.blocking import BlockingScheduler
def scrape_bbc_headlines():
 url = 'https://www.bbc.com/news' # URL of the BBC News website
 response = requests.get(url)
 if response.status_code == 200: # Check if the request was successful
 soup = BeautifulSoup(response.text, 'html.parser')
 # Find elements that contain headlines (specific to the BBC structure)
 headlines = soup.find_all('h3', class_='gs-c-promo-heading__title')
 # Extract text from the headline elements
 headline_texts = [headline.get_text(strip=True) for headline in headlines]
 return headline_texts
 else:
 print("Failed to retrieve data")
# Test the function
headlines = scrape_bbc_headlines()
if headlines:
 for idx, headline in enumerate(headlines, start=1):
 print(f"Headline {idx}: {headline}")
else:
 print("No headlines found")
# Create a scheduler
scheduler = BlockingScheduler()
# Schedule the scraping function to run every hour (you can change the
interval as needed)
scheduler.add_job(scrape_bbc_headlines, 'interval', hours=1)
try:
 # Start the scheduler
 scheduler.start()
except KeyboardInterrupt:
 # If you want to stop the scheduler manually (Ctrl+C)
 pass
```

Data Processed:

Here is a generic breakdown of the steps involved in web scraping and subsequent data processing:

- **Import Libraries:** Begin by importing the necessary libraries for web scraping. Common libraries include BeautifulSoup or Scrapy for HTML parsing and requests for making HTTP requests.

```
from bs4 import BeautifulSoup
import requests
```

- **Send HTTP Request:** Use the **requests** library to send an HTTP request to the target website and retrieve the HTML content.

```
url = "https://example.com"
response = requests.get(url)
```

- **Parse HTML Content:** Use BeautifulSoup or a similar library to parse the HTML content obtained from the website.

```
soup = BeautifulSoup(response.text, 'html.parser')
```

- **Locate Data:** Identify the HTML elements containing the data you want to scrape. Use CSS selectors or Xpath to locate specific elements.

```
headlines = soup.select('.headline')
```

- **Repeat as Necessary:** If scraping multiple pages or websites, encapsulate the scraping logic in a loop. Be mindful of the website's terms of service and avoid aggressive scraping that may cause issues.

```
for page_number in range(1, 6):
    url = f"https://example.com/page/{page_number}"
```

- **Store or Analyze Data:** Decide whether to store the extracted data in a local file, a database, or analyze it further within your script.

```
with open('headlines.txt', 'w') as file:
    for headline in cleaned_data:
        file.write(headline + '\n')
```

- **Data Cleaning:** Perform any necessary data cleaning or preprocessing steps. This could include removing unnecessary characters, handling missing values, or converting data types.

```
cleaned_data = [clean_text(headline.text) for headline in headlines]
```

Automation:

The BlockingScheduler from the apscheduler package is the main tool used in this work to do automation. Without requiring human input, this scheduler automatically carries out the web scraping operation on a regular basis.
Let's dissect the integration of automation:

- **Initialize the Scheduler:**Create an instance of the **BlockingScheduler** and define how often you want the web scraping function to be executed.

```
scheduler = BlockingScheduler()
```

- **Define the WebScraping Function:**Create a function that encapsulates the web scraping logic you want to automate. This function should perform all the necessary steps, including sending HTTP requests, parsing HTML content, and extracting data.

```
def scrape_data():
```

- **Schedule the Web Scraping Function:**Use the **add_job** method of the scheduler to schedule the web scraping function at regular intervals. Specify the function to be executed and the interval at which it should run.

```
scheduler.add_job(scrape_data, 'interval', hours=1)
```

- **Import the Required Library:** Begin by importing the necessary libraries, including the **BlockingScheduler** from the **apscheduler** library.

```
from apscheduler.schedulers.blocking import BlockingScheduler
```

THANK YOU!!!