Developing Software for Downloading Large Amount of Data from Web Applications Using the Python Programming Language

Q. Asqarov Assistant teacher of the department "Algorithmization and programming technology"

B.Geldibayev Assistant teacher of the department "Algorithmization and programming technology"

ABSTRACT

In this article explores software development for downloading large amounts of data in web applications using the beautiful soup module in the python programming language. Web-scraping is the conversion of large amounts of data into a spreadsheet or API (Application programming interface) using python programming. In the conducted research, was developed software and results were obtained. This software can directly access data in web applications using Hypertext Transfer Protocol or a browser.

Keywords: Python, web scraping, API, beautiful soup module

Introduction:

Today, most web sites consist of a very large database. These include stock exchange prices, product prices in online stores, and information about employees and students in educational institutions of offices and organizations. If you want to download this information, you will have to create a new document and copy and paste each information into the document. To solve this problem, it is now possible to parse the web application through the python programming language.

Web scraping is an automatic method of extracting large amounts of data from websites. Most of this data is unstructured data in HTML format that is then converted into structured data in a spreadsheet or database for use in various applications. There are different ways to perform web scraping to extract data from websites. These include using online services, custom APIs, or even creating your own Web scraping code from scratch. Many large websites like Google, Twitter, Facebook, StackOverflow, etc. have APIs that allow you to access their data in a structured format. This is the best option, but there are other sites that do not allow users to access large amounts of data in a structured form or are not technologically advanced. In this case, it is effective to use web scraping to scrape the website for information.

Research methodology and results:

Web scraping requires two parts. A web crawler is an artificial intelligence algorithm that crawls web applications in search of relevant information by following links on the web. On the other hand, scraping is a special tool designed to extract data from a website. The

design of the parser can vary depending on the complexity and scope of the project so that it can extract data quickly and accurately.

Installing required libraries:

The easiest way to install external libraries in Python is to use pip. pip is a package management system used to install and manage software packages written in Python (pic-1).

```
pip install requests
pip install html5lib
pip install bs4
```

Accessing HTML content from a web page:

```
import requests
URL = " https://www.geeksforgeeks.org/data-structures/ "
r = requests.get(URL)
print(r.content)
```

Let's analyze this code in sequence:

- The requests library is imported first.
- Then enter the url address of the site we want to parse.
- An http request is sent to the specified url, and the response received from the server is stored in the r variable.
- At the end, the html content of the page is printed.

Scraping HTML content:

```
import requests
from bs4 import BeautifulSoup

URL = " http://www.values.com/inspirational-quotes "
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib')
# If an error occurs with the htm5lib library, import the library using import html5lib or install it in the console using pip install html5lib
print(soup.prettify())
```

What's really cool about the BeautifulSoup library is that it's built on top of HTML scraping libraries like html5lib, lxml, html.parser, and more. In this way, the BeautifulSoup object and the parser library can be created at the same time. In the example above,

```
soup = BeautifulSoup(r.content, 'html5lib')
```

We create a BeautifulSoup object by passing two arguments:

- r.content : This is the raw HTML content.
- html5lib : Specifying the HTML parser we want to use.

At the end of the program, soup.prettify() is printed, which provides a visual representation of the parse tree generated from the raw HTML.

Searching and navigating the analysis tree

Now we want to extract some useful information from the HTML content. The soup object contains all the data in an internal structure that can be extracted programmatically. In our example, we're scraping a web page with some quotes. So we want to create a program to store these quotes (and all the related information about them).

```
# The program code parses the website
#and then saves it in csv format
```

```
import requests
from bs4 import BeautifulSoup
import csv

URL = " http://www.values.com/inspirational-quotes "
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib')

quotes=[] # list of quote stories

table = soup.find('div', attrs = {'id':'all_quotes'})

for row in table.findAll('div',
attrs = {'class':'col-6 col-lg-3 text-center margin-30px-bottom cm-margin-30px-top'}):
quote = {}
quote['theme'] = row.h5.text
quote['url'] = row.a['href']
quote['img'] = row.img['src']
quote['lines'] = row.img['alt'].split(" #")[0]
quote['author'] = row.img['alt'].split(" #")[1]
quotes.append(quote)

filename = 'inspirational_quotes.csv'
with open(filename, 'w', newline='') as f:
w = csv.DictWriter(f,['theme','url','img','lines','author'])
w.writeheader()
for quote in quotes:
w.writerow(quote )
```

Notice that all quotes are inside a div container with id "all_quotes". So, we find the div element (called table in the above code) using the find() method:

```
table = soup.find('div', attrs = {'id':'all_quotes'})
```

The first argument is the HTML tag you want to search for, and the second argument is a dictionary type element to specify additional attributes associated with that tag. The find() method returns the first matching element. You can try printing table.prettify() to understand what this piece of code does.

Now, you can see that every quote class in the table element is inside a quote div container. So we iterate through each div container whose class is a quote. Here we use the findAll() method, which is similar to the find by arguments method, but it returns a list of all matching elements. Each quote is now repeated using a variable called string. Here is an example of HTML content for a better understanding :

```
<div class="row" id="all_quotes">
 <div class="col-6 col-lg-3 text-center margin-30px-bottom sm-margin-30px-top">
  <a href="/inspirational-quotes/8052-never-mistake-knowledge-for-wisdom-one-helps">
   <img alt="Never mistake knowledge for wisdom. One helps you make a living; the other helps you make a life. #<Author:0x00007f1917948248>"
   class="margin-10px-bottom shadow" src="https://assets.passiton.com/quotes/quote_artwork/8052/medium/20200324_tuesday_quote.jpg?1584726475"
   width="310" height="310" data-no-retina>
  </a>
  <h5 class="value_on_red">
   <a href="/inspirational-quotes/8052-never-mistake-knowledge-for-wisdom-one-helps">WISDOM</a>
  </h5>
 </div>
```

We create a dictionary to store all the information about the quote. An embedded structure can be accessed using the dot symbol:

```
for row in table.find_all_next('div', attrs = {'class': 'col-6 col- lg-3 text-center margin-30px-bottom cm-margin-30px-top'}):
quote = {}
quote['theme'] = row.h5.text
quote['url'] = row.a['href']
quote['img'] = row.img['src']
quote['lines'] = row.img['alt'].split(" #")[0]
quote['author'] = row.img['alt'].split(" #")[1]
quotes.append(quote)
```

To access the text inside an HTML element, we use .text:

```
quote['theme'] = row.h5.text
```

We can add, delete, modify and access tag attributes. This is done by treating the tag as a dictionary:

```
quote['url'] = row.a['href']
```

Finally, all the quotes are appended to the list called quotes.

Finally, we would like to save all our data in some CSV file.

We can add, delete, modify and access tag attributes. This is done by treating the tag as a dictionary:

```
filename = 'inspirational_quotes.csv'
with open(filename, 'w', newline='') as f:
w = csv.DictWriter(f,['theme','url','img','lines','author'])
w.writeheader()
for quote in quotes:
w.writerow(quote)
```

Here we will create a CSV file named inspirational_quotes.csv and save all the quotes in it for later use.

Summary:

In this article, we looked at what web scraping is, how it is used, and what the process involves. The main interpretations include:

- Web scraping can be used to collect all types of data: From images to videos, text, digital data, and more.
- Web crawling has multiple purposes: From social media trolling for contact scraping and brand mentions to SEO audits, the possibilities are endless.

Taking the time to plan what needs to be parsed in advance will save you effort in the long run when it comes to cleaning up your data. Python is a popular tool for scraping web applications: Python libraries such as Beautifulsoup, scrapy, and pandas are common tools for scraping web applications.

Before scraping web applications, you should check the laws in different jurisdictions and be careful not to violate the site's terms of service.

References:

1. "Web scraping using Python". Ryan Mitchell. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472
2. Izuchaem Python . Programmirovanie igr, visualization dannyx, web prilogenia. Eric Metz. — SPb.: Peter, 2017. — 496 p.: il. — (Series "Biblioteka programmer").
3. Programming Python. Mark Lutz. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.