Web Scraping with Python

Long Lin & Shiang Xuanyuan

Introduction

Description

There is so much information on the Internet that a human being can't master it all in a lifetime. What we need to do is not access to this information but using an extensible way to collect, organize, and analyze it. Imagine you have to pull a large amount of data from websites and you want to do it as quickly as possible. How would you do it without manually going to each website and getting the data? Well, "Web Scraping" is the answer. Web Scraping just makes this job easier and faster.

Although we could use web scraping in R, the most common and easiest way of retrieving data from internet is using Python. We would give a brief introduction on how to use Python to do the web scraping, storing data into a csv file, then we could use any another languages to do the data analysis, data visualization, etc.

Why is Web Scraping Used?

Web scraping is used to collect large information from websites. But why does someone have to collect such large data from websites? To know about this, let's look at the applications of web scraping:

- **Price Comparison:** Services such as ParseHub use web scraping to collect data from online shopping websites and use it to compare the prices of products.
- **Email address gathering:** Many companies that use email as a medium for marketing, use web scraping to collect email ID and then send bulk emails.
- **Social Media Scraping:** Web scraping is used to collect data from Social Media websites such as Twitter to find out what's trending.
- **Research and Development:** Web scraping is used to collect a large set of data (Statistics, General Information, Temperature, etc.) from websites, which are analyzed and used to carry out Surveys or for R&D.
- **Job listings:** Details regarding job openings, interviews are collected from different websites and then listed in one place so that it is easily accessible to the user.

What is Web Scraping?

Web scraping is an automated method used to extract large amounts of data from websites. The data on the websites are unstructured. Web scraping helps collect these unstructured data and store it in a structured form. There are different ways to scrape websites such as online Services, APIs or writing your own code.

How to Scrape Data From A Website?

When you run the code for web scraping, a request is sent to the URL that you have mentioned. As a response to the request, the server sends the data and allows you to read the HTML or XML page. The code then, parses the HTML or XML page, finds the data and extracts it.

To extract data using web scraping with python, you need to follow these basic steps:

1. Find the URL that you want to scrape
2. Inspecting the Page
3. Find the data you want to extract
4. Write the code
5. Run the code and extract the data
6. Store the data in the required format

Note: Before we extract data from a website, it would be better to check whether a website allows web scraping or not, you can look at the website's "robots.txt" file. You can find this file by appending "/robots.txt" to the URL that you want to scrape.

Libraries used for Web Scraping

As we know, Python is has various applications and there are different libraries for different purposes. In our further demonstration, we will be using the following libraries:

- **Requests:** the tool of website downloading
- **BeautifulSoup:** Beautiful Soup is a Python package for parsing HTML and XML documents. It creates parse trees that is helpful to extract the data easily.
- **Pandas:** Pandas is a library used for data manipulation and analysis. It is used to extract the data and store it in the desired format.

There are several libraries may also be used in web scraping:

- **Re:** regular expression, used to data cleaning and organizing.
- **Selenium:** Selenium is a web testing library. It is used to automate browser activities.

Get started with scraping

HTML Format

Before we move the coding part, it would be necessarily to understand the formats of HTML and some rules of scraping. For example, the following is a piece of a html document:

```
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and
their names were
<a href="http://www.jb51.net" class="sister" id="link1">Elsie</a>,
<a href="http://www.jb51.net" class="sister" id="link2">Lacie</a> and
<a href="http://www.jb51.net" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
```

Note that every services a part of the website:

1. HTML documents are stored between `<html>` and `</html>`.
2. The meta statements and script statements must be between `<head>` and `</head>`.
3. The visible parts of HTML documents are between `<body>` and `</body>`.
4. `<p>` is defined as the paragraph of article.
5. `<a>` indicates the hyperlink.
6. Attribute - `id` : is unique for every HTML tag, and the array in `id` must be unique in the entire HTML document.
7. Attribute - `class` : is used to style labels with the same value.
8. We would use `id` and `class` to locate the information we want.

Web Scraping Examples:

The Basics of Web Scraping with BeautifulSoup

Part 1: Loading Web Pages with 'request'

The requests module allows you to send HTTP requests using Python.

The HTTP request returns a Response Object with all the response data (content, encoding, status, and so on). One example of getting the HTML of a page this link.

In [30]:
```
import requests

# Print all outcomes
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"


# Make a request to URL
# Store the result in 'res' variable
res = requests.get('https://codedamn.com')

# Store the text response in a variable called txt
txt = res.text
# Store the status code in a variable called status
status = res.status_code
```

```
# Print the first five line
line_nu = 0
for line in res:
    if line_nu<5:
        print(line.strip())
        line_nu += 1
    else:
        break

#print(res.text)
#print(res.status_code)
```

```
b'<!DOCTYPE html><html lang="en" class="text-gray-500 antialiased bg-white js-focus-visib
le"><head><style>@font-face {\n    font-fa'
b'mily: Inter var;\n    font-weight: 100 900;\n    font-display: swap;\n    font-style: n
ormal;\n    font-named-instance:"Regular";\n\tf'
b'ont-feature-settings: "cv02","cv03","cv04","cv11";\n    src: url(/assets/fonts/inter-va
r.woff2) format("woff2")\n}\n\n@font-face {'
b'font-family: Inter var;\n    font-weight: 100 900;\n    font-display: swap;\n    font-s
tyle: italic;\n    font-named-instance:"It'
b'alic";\n\tfont-feature-settings: "cv02","cv03","cv04","cv11";\n    src: url(/assets/fon
ts/inter-var2.woff2) format("woff2")\n}\n</sty'
```

Part 2: Extracting title with BeautifulSoup

Some features that make BeautifulSoup a powerful solution are:

1. It provides a lot of simple methods and Pythonic idioms for navigating, searching, and modifying a DOM tree. It doesn't take much code to write an application
2. Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

Basically, BeautifulSoup can parse anything on the web you give it.

Here's a simple example of BeautifulSoup (getting the HTML of a page this link):

In [21]:
```
import requests
from bs4 import BeautifulSoup

# Make a request to the goal URL
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/"
soup = BeautifulSoup(page.content, 'html.parser')

# Use BeautifulSoup to store the title of this page into a variable called page_title
# Extract title of page
page_title = soup.title.text

# print the result
print(page_title)
```

```
codedamn Web Scraper demo
```

Part 3: Soup-ed body and head

Store body content (without calling .text) of URL in page_body

Store head content (without calling .text) of URL in page_head

When you try to print the page_body or page_head you'll see that those are printed as strings. But in reality, when you print(type page_body) you'll see it is not a string but it works fine.

In [22]:
```
import requests
from bs4 import BeautifulSoup

# Make a request
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/"
soup = BeautifulSoup(page.content, 'html.parser')

# Extract title of page
page_title = soup.title

# Extract body of page
page_body = soup.body

# Extract head of page
page_head = soup.head

# print the result
print(page_title, page_head)
```

```
<title>codedamn Web Scraper demo</title> <head>
<!-- Anti-flicker snippet (recommended)  -->
<style>
                    .async-hide {
                            opacity: 0 !important;
                    }
                </style>
<title>codedamn Web Scraper demo</title>
<meta charset="utf-8"/>
<meta content="IE=edge,chrome=1" http-equiv="X-UA-Compatible"/>
<meta content="web scraping,Web Scraper,Chrome extension,Crawling,Cross platform scraper,
" name="keywords"/>
<meta content="The most popular web scraping website." name="description"/>
<link href="/webscraper-python-codedamn-classroom-website/favicon.png" rel="icon" sizes
="128x128"/>
<meta content="width=device-width, initial-scale=1.0" name="viewport"/>
<link href="/webscraper-python-codedamn-classroom-website/app.css" rel="stylesheet"/>
<link href="/webscraper-python-codedamn-classroom-website/logo-icon.png" rel="apple-touch
-icon"/>
<script defer="" src="/webscraper-python-codedamn-classroom-website/app.js"></script>
</head>
```

Part 4: select with BeautifulSoup

Now that we have explored some parts of BeautifulSoup, let's look how we can select DOM elements with BeautifulSoup methods.

Once we have the `soup` variable (like previous labs), we can work with `.select` on it which is a CSS selector inside BeautifulSoup. That is, you can reach down the DOM tree just like how we will select elements with CSS. `.select` returns a Python list of all the elements. This is why we selected only the first element here with the `[0]` index. Let's look at an example:

In [24]:
```
import requests
```

```
from bs4 import BeautifulSoup
# Make a request
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/"
soup = BeautifulSoup(page.content, 'html.parser')

# Create all_h1_tags as empty list
all_h1_tags = []

# Set all_h1_tags to all h1 tags of the soup
for element in soup.select('h1'):
    all_h1_tags.append(element.text)

# Create seventh_p_text and set it to 7th p element text of the page
seventh_p_text = soup.select('p')[6].text

print(all_h1_tags, seventh_p_text)
```

['Test Sites', 'E-commerce training site'] 7 reviews

Part 5: Top items being scraped right now

Let's go ahead and extract the top items scraped from the URL: https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/

If we open this page in a new tab, we'll see some top items.Now we need to scrape out their names and store them in a list called `top_items`. We will also extract out the reviews for these items as well.

In the following process:

1. First of all we select all the `div.thumbnail` elements which gives us a list of individual products
2. Then we iterate over them
3. Because `select` allows us to chain over itself, we can use select again to get the title.
4. Note that because we're running inside a loop for `div.thumbnail` already, the `h4 > a.title` selector would only give us one result, inside a list. We select that list's 0th element and extract out the text.
5. Finally we strip any extra whitespace and append it to our list.

In [26]:
```
import requests
from bs4 import BeautifulSoup
# Make a request
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/"
soup = BeautifulSoup(page.content, 'html.parser')

# Create top_items as empty list
top_items = []

# Extract and store in top_items according to instructions on the left
products = soup.select('div.thumbnail')
for elem in products:
    title = elem.select('h4 > a.title')[0].text
    review_label = elem.select('div.ratings')[0].text
```

```
    info = {
        "title": title.strip(),
        "review": review_label.strip()
    }
    top_items.append(info)

print(top_items)
```

```
[{'title': 'Asus AsusPro Adv...', 'review': '7 reviews'}, {'title': 'Asus ROG Strix
G...', 'review': '4 reviews'}, {'title': 'Acer Aspire 3 A3...', 'review': '2 reviews'}]
```

Part 6: Extracting Links

So far we have seen how we can extract the text, or rather innerText of elements. Let's now see how we can extract attributes by extracting links from the page.

Here, we extract the `href` attribute just like you did in the image case. The only thing we're doing is also checking if it is None. We want to set it to empty string, otherwise we want to strip the whitespace.

In [27]:
```
import requests
from bs4 import BeautifulSoup
# Make a request
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/"
soup = BeautifulSoup(page.content, 'html.parser')

# Create top_items as empty list
all_links = []

# Extract and store in top_items according to instructions on the left
links = soup.select('a')
for ahref in links:
    text = ahref.text
    text = text.strip() if text is not None else ''

    href = ahref.get('href')
    href = href.strip() if href is not None else ''
    all_links.append({"href": href, "text": text})

print(all_links)
```

```
[{'href': '', 'text': 'Toggle navigation'}, {'href': '/webscraper-python-codedamn-classro
om-website/', 'text': ''}, {'href': '#page-top', 'text': ''}, {'href': '/webscraper-pytho
n-codedamn-classroom-website/', 'text': 'Web Scraper'}, {'href': '/webscraper-python-code
damn-classroom-website/cloud-scraper', 'text': 'Cloud Scraper'}, {'href': '/webscraper-py
thon-codedamn-classroom-website/pricing', 'text': 'Pricing'}, {'href': '#section3', 'tex
t': 'Learn'}, {'href': '/webscraper-python-codedamn-classroom-website/documentation', 'te
xt': 'Documentation'}, {'href': '/webscraper-python-codedamn-classroom-website/tutorial
s', 'text': 'Video Tutorials'}, {'href': '/webscraper-python-codedamn-classroom-website/h
ow-to-videos', 'text': 'How to'}, {'href': '/webscraper-python-codedamn-classroom-websit
e/test-sites', 'text': 'Test Sites'}, {'href': 'https://forum.webscraper.io/', 'text': 'F
orum'}, {'href': 'https://chrome.google.com/webstore/detail/web-scraper/jnhgnonknehpejjne
hehllkliplmbmhn?hl=en', 'text': 'Install'}, {'href': 'https://cloud.webscraper.io/', 'tex
t': 'Login'}, {'href': '/webscraper-python-codedamn-classroom-website/test-sites/e-commer
ce/allinone', 'text': 'Home'}, {'href': '/webscraper-python-codedamn-classroom-website/te
st-sites/e-commerce/allinone/computers', 'text': 'Computers'}, {'href': '/webscraper-pyth
on-codedamn-classroom-website/test-sites/e-commerce/allinone/phones', 'text': 'Phones'},
{'href': '/webscraper-python-codedamn-classroom-website/test-sites/e-commerce/allinone/pr
oduct/593', 'text': 'Asus AsusPro Adv...'}, {'href': '/webscraper-python-codedamn-classro
```

om-website/test-sites/e-commerce/allinone/product/583', 'text': 'Asus ROG Strix G...'}, {'href': '/webscraper-python-codedamn-classroom-website/test-sites/e-commerce/allinone/product/576', 'text': 'Acer Aspire 3 A3...'}, {'href': '/webscraper-python-codedamn-classroom-website/', 'text': 'Web Scraper browser extension'}, {'href': '/webscraper-python-codedamn-classroom-website/pricing', 'text': 'Web Scraper Cloud'}, {'href': '/webscraper-python-codedamn-classroom-website/contact', 'text': 'Contact'}, {'href': '/webscraper-python-codedamn-classroom-website/privacy-policy', 'text': 'Website Privacy Policy'}, {'href': '/webscraper-python-codedamn-classroom-website/extension-privacy-policy', 'text': 'Browser Extension Privacy Policy'}, {'href': 'http://webscraperio.us-east-1.elasticbeanstalk.com/downloads/Web_Scraper_Media_Kit.zip', 'text': 'Media kit'}, {'href': '/webscraper-python-codedamn-classroom-website/jobs', 'text': 'Jobs'}, {'href': '/webscraper-python-codedamn-classroom-website/blog', 'text': 'Blog'}, {'href': '/webscraper-python-codedamn-classroom-website/documentation', 'text': 'Documentation'}, {'href': '/webscraper-python-codedamn-classroom-website/tutorials', 'text': 'Video Tutorials'}, {'href': '/webscraper-python-codedamn-classroom-website/screenshots', 'text': 'Screenshots'}, {'href': '/webscraper-python-codedamn-classroom-website/test-sites', 'text': 'Test Sites'}, {'href': 'https://forum.webscraper.io/', 'text': 'Forum'}, {'href': 'mailto:info@webscraper.io', 'text': 'info@webscraper.io'}, {'href': 'https://www.facebook.com/webscraperio/', 'text': ''}, {'href': 'https://twitter.com/webscraperio', 'text': ''}, {'href': '#', 'text': 'Web Scraper'}]

Part 7: Generating CSV from data

Finally, let's understand how we can generate CSV from a set of data. We will create a CSV with the following headings:

1. Product Name
2. Price
3. Description
4. Reviews
5. Product Image These products are located in the `div.thumbnail` .

- Product Name is the whitespace trimmed version of the name of the item (example - Asus AsusPro Adv..)
- Price is the whitespace trimmed but full price label of the product (example - 1101.83)
- The description is the whitespace trimmed version of the product description (example - Asus AsusPro Advanced BU401LA-FA271G Dark Grey, 14", Core i5-4210U, 4GB, 128GB SSD, Win7 Pro)
- Reviews are the whitespace trimmed version of the product (example - 7 reviews)
- Product image is the URL (src attribute) of the image for a product (example - /webscraper-python-codedamn-classroom-website/cart2.png)
- The name of the CSV file should be products.csv and should be stored in the same directory as our script.py file

In [28]:
```
import requests
from bs4 import BeautifulSoup
import csv
# Make a request
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/"
soup = BeautifulSoup(page.content, 'html.parser')

# Create top_items as empty list
all_products = []
```

```
# Extract and store in top_items according to instructions on the left
products = soup.select('div.thumbnail')
for product in products:
    name = product.select('h4 > a')[0].text.strip()
    description = product.select('p.description')[0].text.strip()
    price = product.select('h4.price')[0].text.strip()
    reviews = product.select('div.ratings')[0].text.strip()
    image = product.select('img')[0].get('src')

    all_products.append({
        "name": name,
        "description": description,
        "price": price,
        "reviews": reviews,
        "image": image
    })


keys = all_products[0].keys()

with open('products.csv', 'w', newline='') as output_file:
    dict_writer = csv.DictWriter(output_file, keys)
    dict_writer.writeheader()
    dict_writer.writerows(all_products)
```

Out[28]: 38

The `for` block is the most interesting here. We extract all the elements and attributes from what we've learned so far in all the labs.

When we run this code, we end up with a nice CSV file.

Part 8: Present the result of CSV from Part7:

In [36]:
```
import pandas as pd
data = pd.read_csv("products.csv")
data
```

Out[36]:

	name	description	price	reviews	image
0	Asus AsusPro Adv...	Asus AsusPro Advanced BU401LA-FA271G Dark Grey...	$1139.54	7 reviews	/webscraper-python-codedamn-classroom-website/...
1	Asus ROG Strix G...	Apple MacBook Air 13.3", Core i5 1.8GHz, 8GB, ...	$1101.83	4 reviews	/webscraper-python-codedamn-classroom-website/...
2	Acer Aspire 3 A3...	Acer Aspire 3 A315-51 Black, 15.6" FHD, Core\n...	$494.71	2 reviews	/webscraper-python-codedamn-classroom-website/...

Why is Python Good for Web Scraping?

Here is the list of features of Python which makes it more suitable for web scraping.

- **Ease of Use:** Python is simple to code. You do not have to add semi-colons ";" or curly-braces "{}" anywhere. This makes it less messy and easy to use.

- **Large Collection of Libraries:** Python has a huge collection of libraries such as Numpy, Matlplotlib, Pandas etc., which provides methods and services for various purposes. Hence, it is suitable for web scraping and for further manipulation of extracted data.
- **Dynamically typed:** In Python, you don't have to define datatypes for variables, you can directly use the variables wherever required. This saves time and makes your job faster.
- **Easily Understandable Syntax:** Python syntax is easily understandable mainly because reading a Python code is very similar to reading a statement in English. It is expressive and easily readable, and the indentation used in Python also helps the user to differentiate between different scope/blocks in the code.
- **Small code, large task:** Web scraping is used to save time. But what's the use if you spend more time writing the code? Well, you don't have to. In Python, you can write small codes to do large tasks. Hence, you save time even while writing the code.
- **Community:** What if you get stuck while writing the code? You don't have to worry. Python community has one of the biggest and most active communities, where you can seek help from.

More Techniques for Web Scraping

BeautifulSoup is beginner-friendly and well-documented, the size of web scraping project supported can be large or small. However, there still have several options for web scraping:

1. **Scrapy:** Scrapy is a fast high-level web crawling and web scraping framework, used to crawl websites and extract structured data from their pages.
2. **Selenium:** simulate a request in the browser and wait for the dynamic content to be displayed, by using selenium package in Python, the path of ChromeDriver is required.
3. Try to integrate some public APIs into code, Data retrieval is much more efficient than page crawling.
4. When the size of dataset is too large, consider using **MySQL** or **MongoDB** to store the data.

Referrences:

1. https://www.edureka.co/blog/web-scraping-with-python/
2. https://www.freecodecamp.org/news/how-to-scrape-websites-with-python-2/
3. https://www.freecodecamp.org/news/web-scraping-python-tutorial-how-to-scrape-data-from-a-website/
4. https://oxylabs.io/blog/python-web-scraping
5. https://gist.github.com/gjreda/f3e6875f869779ec03db