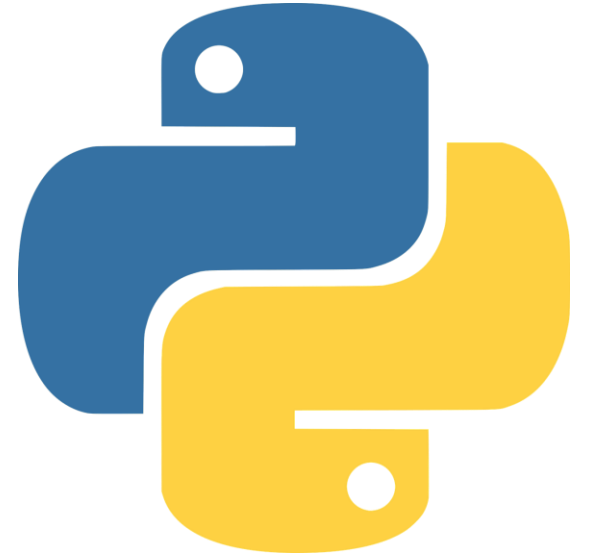


Web Scraping with Python Using BeautifulSoup

In this tutorial, we'll show you how to perform web scraping using Python and the BeautifulSoup library.



BeautifulSoup

How Does Web Scraping Work?

When we scrape the web, we write code that sends a request to the server that's hosting the page we specified. The server will return the source code — HTML, mostly — for the page (or pages) we requested.

So far, we're essentially doing the same thing a web browser does — sending a server request with a specific URL and asking the server to return the code for that page.

But *unlike* a web browser, our web scraping code won't interpret the page's source code and display the page visually. Instead, we'll write some custom code that filters through the page's source code looking for specific elements we've specified, and extracting whatever content we've instructed it to extract.

For example, if we wanted to get all of the data from inside a table that was displayed on a web page, our code would be written to go through these steps in sequence:

1. Request the content (source code) of a specific URL from the server
2. Download the content that is returned
3. Identify the elements of the page that are part of the table we want
4. Extract and (if necessary) reformat those elements into a dataset we can analyze or use in whatever way we require.

If that all sounds very complicated, don't worry! Python and BeautifulSoup have built-in features designed to make this relatively straightforward.

One thing that's important to note: from a server's perspective, requesting a page via web scraping is the same as loading it in a web browser. When we use code to submit these requests, we might be "loading" pages much faster than a regular user, and thus quickly eating up the website owner's server resources.

Why Use Python for Web Scraping?

As previously mentioned, it's possible to do web scraping with many programming languages.

However, one of the most popular approaches is to use Python and the Beautiful Soup library, as we'll do in this tutorial.

Learning to do this with Python will mean that there are lots of tutorials, how-to videos, and bits of example code out there to help you deepen your knowledge once you've mastered the Beautiful Soup basics.

Is Web Scraping Legal?

Unfortunately, there's not a cut-and-dry answer here. Some websites explicitly allow web scraping. Others explicitly forbid it. Many websites don't offer any clear guidance one way or the other.

Before scraping any website, we should look for a terms and conditions page to see if there are explicit rules about scraping. If there are, we should follow them. If there are not, then it becomes more of a judgement call.

Remember, though, that *web scraping consumes server resources for the host website*. If we're just scraping one page once, that isn't going to cause a problem. But if our code is scraping 1,000 pages once every ten minutes, that could quickly get expensive for the website owner.

Web Scraping Best Practices:

Never scrape more frequently than you need to.

Consider caching the content you scrape so that it's only downloaded once.

Build pauses into your code using functions like `time.sleep()` to keep from overwhelming servers with too many requests too quickly.

Beautiful Soup 4 Setup

[Beautiful Soup](#) is a Python library for parsing structured data. It allows you to interact with HTML in a similar way to how you interact with a web page using developer tools. The library exposes a couple of intuitive functions you can use to explore the HTML you received. To get started, use your terminal to install Beautiful Soup:

```
python -m pip install beautifulsoup4
```

Reading HTML Files

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>A simple example page</title>
</head>
<body>
  <p>Here is some simple content for this page.</p>
</body>
</html>
```

Reading HTML Files

We first have to import the library, and create an instance of the BeautifulSoup class to parse our document

Then we open file html

Then we can print out the HTML content of the page, formatted nicely, using the prettify method on the BeautifulSoup object.

`web_scraping.py`

```
from bs4 import BeautifulSoup

# HTML From File
with open("index.html", "r") as f:
    doc = BeautifulSoup(f, "html.parser")

print(doc.prettify())
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      A simple example page
    </title>
  </head>
  <body>
    <p>
      Here is some simple content for this page.
    </p>
  </body>
</html>
```


Find By Tag Name

web_scraping.py

```
from bs4 import BeautifulSoup

# HTML From File
with open("index.html", "r") as f:
    doc = BeautifulSoup(f, "html.parser")

tag = doc.title

print(tag)
```

```
<title>A simple example page</title>
```

```
Process finished with exit code 0
```

web_scraping.py

```
from bs4 import BeautifulSoup

# HTML From File
with open("index.html", "r") as f:
    doc = BeautifulSoup(f, "html.parser")

tag = doc.title

print(tag.string)
```

```
A simple example page
```

```
Process finished with exit code 0
```

Find By Tag Name

We can also modify these tags

web_scraping.py

```
from bs4 import BeautifulSoup

# HTML From File
with open("index.html", "r") as f:
    doc = BeautifulSoup(f, "html.parser")

tag = doc.title
tag.string = 'hello'

print(tag)
```

```
<title>hello</title>
```

```
Process finished with exit code 0
```

web_scraping.py

```
from bs4 import BeautifulSoup

# HTML From File
with open("index.html", "r") as f:
    doc = BeautifulSoup(f, "html.parser")

tag = doc.title
tag.string = 'hello'

print(doc.prettify())
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      hello
    </title>
  </head>
  <body>
    <p>
      Here is some simple content for this page.
    </p>
  </body>
</html>
```

```
Process finished with exit code 0
```

Finding all instances of a tag at once

If we want to extract a single tag, we can use the `find_all` method, which will find all the instances of a tag on a page.

```
from bs4 import BeautifulSoup

# HTML From File
with open("index.html", "r") as f:
    doc = BeautifulSoup(f, "html.parser")

tags = doc.find_all('p')
print(tags)
```

[web_scraping.py](#)

Note that `find_all` returns a list, so we'll have to loop through, or use list indexing, it to extract text:

```
from bs4 import BeautifulSoup

# HTML From File
with open("index.html", "r") as f:
    doc = BeautifulSoup(f, "html.parser")

tags = doc.find_all('p')[0].get_text()
print(tags)
```

[web_scraping.py](#)

If you instead only want to find the first instance of a tag, you can use the `find` method, which will return a single BeautifulSoup object

Parsing Website HTML

Now that we understand the structure of a web page, it's time to get into the fun part: scraping the content we want!

The first thing we'll need to do to scrape a web page is to download the page. We can download pages using the Python [requests](#) library.

The requests library will make a GET request to a web server, which will download the HTML contents of a given web page for us. There are several different types of requests we can make using requests, of which GET is just one.


Type the following command in your terminal to install the external requests library:

```
python -m pip install requests
```

Parsing Website HTML

So, I want to check the price of the specific GPU:

https://www.newegg.ca/gigabyte-geforce-rtx-3080-ti-gv-n308tgaming-oc-12gd/p/N82E16814932436?Description=3080&cm_re=3080-_-14-932-436-_-Product



GIGABYTE™

GIGABYTE Gaming GeForce RTX 3080 Ti 12GB GDDR6X PCI Express 4.0 ATX Video Card GV-N308TGAMING OC-12GD


★★★★★ (47)
📺 (13)
[Write a Review](#)

🔍 See more "rtx 3080 ti"
 [SHARE](#)

In stock.

[Ships from China.](#)

Fees for brokerage and duty included in price. Most customers receive within **5-32 days**.



PC BUILDER
Choosing Parts Made Easy

[Start Building ▶](#)

- 12GB 384-Bit GDDR6X
- 2 x HDMI 3 x DisplayPort
- PCI Express 4.0

4 New from **\$2,881.18**

Sold and shipped by: **Sennow** 🇨🇦

127 Ratings (55% Positive)

~~\$2,910.56~~
\$2,881.18
 \$50.00 Shipping

1 [+](#) [-](#) [ADD TO CART ▶](#)

☐ [ADD TO COMPARE](#)
☐ [PRICE ALERT](#)

[♥ ADD TO WISH LIST](#)
 Found on 1 wish list

BEST SELLERS

Millennials Boy

★★★★★ 35 (74% Positive)

\$2,910.29
+ \$49.99 Shipping

[VIEW DETAILS ▶](#)

Corn Electronics.CAN

★★★★★ 2,659 (88% Positive)

\$2,999.99
+ FREE SHIPPING

[VIEW DETAILS ▶](#)

Parsing Website HTML

```
from bs4 import BeautifulSoup
import requests

# HTML From Website
url = "https://www.newegg.ca/gigabyte-geforce-rtx-3080-ti-gv-n308tgaming-oc-12gd/p/N82E16814932436?Description=3080&cm_re=3080-_-14-932-436-_-Product"

result = requests.get(url)
doc = BeautifulSoup(result.text, "html.parser")

print(doc.prettify())
```

web_scraming.py

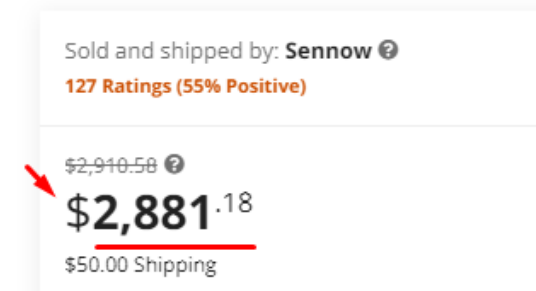
This code issues an HTTP GET request to the given URL. It retrieves the HTML data that the server sends back and stores that data in a Python object.

You successfully fetched the static site content from the Internet!

Locating Text

What I want to do is actually find the price of this GPU

I just want look for the dollar sign and then find the price afterwords



```
from bs4 import BeautifulSoup
import requests

# HTML From Website
url = "https://www.newegg.ca/gigabyte-geforce-rtx-3080-ti-gv-n308tgaming-oc-12gd/p/N82E16814932436?Description=3080&cm_re=3080_-_14-932-436_-_Product"

result = requests.get(url)
doc = BeautifulSoup(result.text, "html.parser")

prices = doc.find_all(text="$")
print(prices)
```

web_scraping.py

Beautiful Soup Tree Structure

web_scraping.py


```
from bs4 import BeautifulSoup
import requests

# HTML From Website
url = "https://www.newegg.ca/gigabyte-geforce-rtx-3080-ti-gv-n308tgaming-oc-12gd/p/N82E16814932436?Description=3080&cm_re=3080_-_14-932-436_-_Product"

result = requests.get(url)
doc = BeautifulSoup(result.text, "html.parser")

prices = doc.find_all(text="$")
parent = prices[0].parent
print(parent.prettify())
```

```
<li class="price-current">
  <span class="price-current-label">
    </span>
    $
    <strong>
      2,881
    </strong>
    <sup>
      .18
    </sup>
  </li>
```



Beautiful Soup Tree Structure

web_scraping.py

```
from bs4 import BeautifulSoup
import requests

# HTML From Website
url = "https://www.newegg.ca/gigabyte-geforce-rtx-3080-ti-gv-n308tgaming-oc-12gd/p/N82E16814932436?Description=3080&cm_re=3080_-_14-932-436_-_Product"

result = requests.get(url)
doc = BeautifulSoup(result.text, "html.parser")

prices = doc.find_all(text="$")
parent = prices[0].parent
strong = parent.find("strong")
print(strong.string)
```

2,881

Process finished with exit code 0