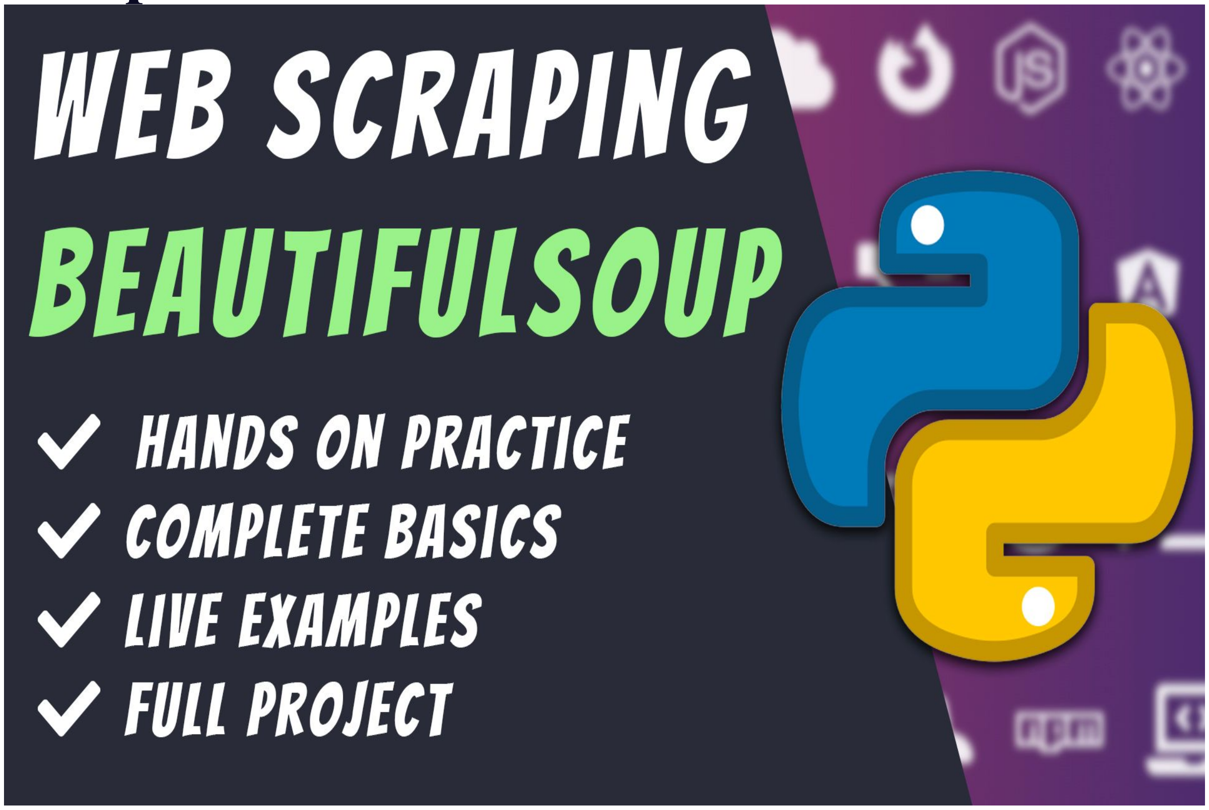
Web Scraping Python Tutorial – How to Scrape Data From A Website



By Mehul Mohan

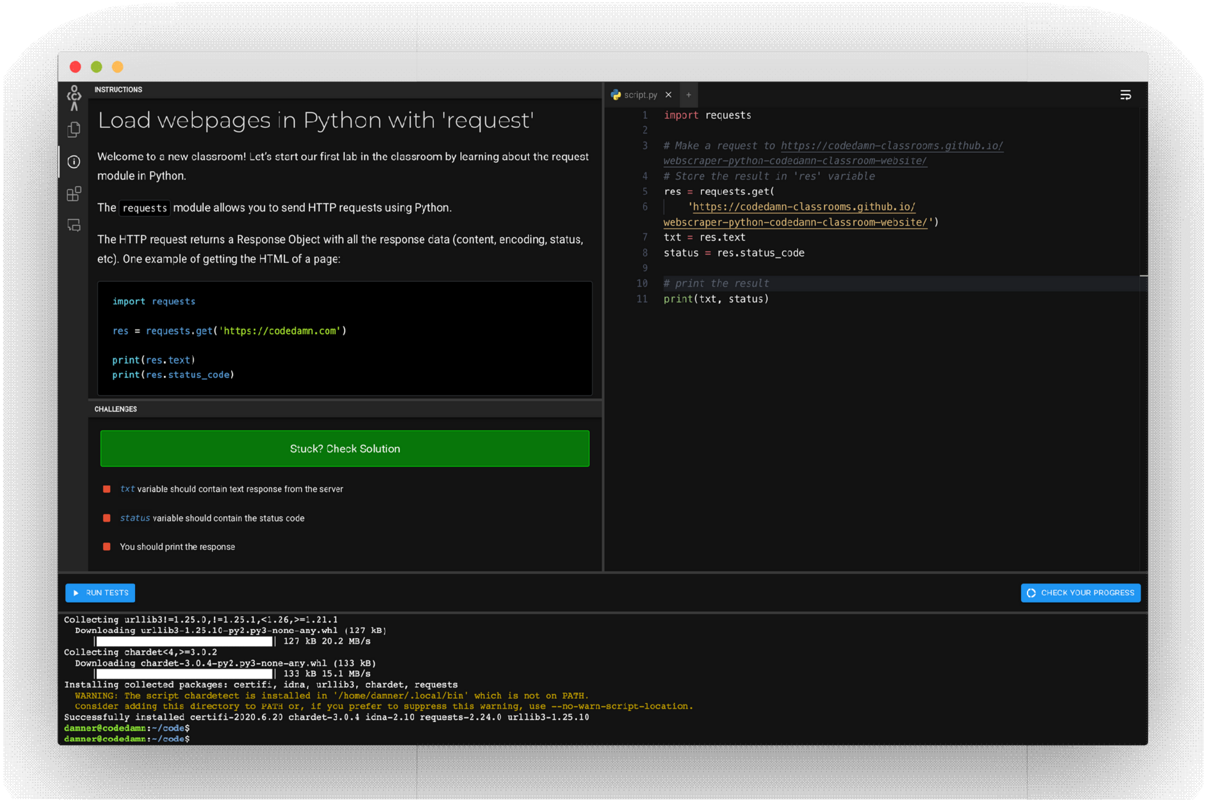
Python is a beautiful language to code in. It has a great package ecosystem, there's much less noise than you'll find in other languages, and it is super easy to use.

Python is used for a number of things, from data analysis to server programming. And one exciting use-case of Python is Web Scraping.

In this article, we will cover how to use Python for web scraping. We'll also work through a complete hands-on classroom guide as we proceed.

Note: We will be scraping a webpage that I host, so we can safely learn scraping on it. Many companies do not allow scraping on their *websites, so this is a good way to learn. Just make sure to check before you scrape.*

Introduction to Web Scraping classroom



Load webpages in Python with 'request'

Welcome to a new classroom! Let's start our first lab in the classroom by learning about the request module in Python.

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, etc). One example of getting the HTML of a page:

```python
import requests

res = requests.get('https://codedamn.com')

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

- 🟥 `txt` variable should contain text response from the server
- 🟥 `status` variable should contain the status code
- 🟥 You should print the response

```python
import requests

# Make a request to https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/
# Store the result in 'res' variable
res = requests.get(
    'https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/')
txt = res.text
status = res.status_code

# print the result
print(txt, status)
```

```
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
  Downloading urllib3-1.25.10-py2.py3-none-any.whl (127 kB)
     |████████████████████████████████| 127 kB 20.2 MB/s
Collecting chardet<4,>=3.0.2
  Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
     |████████████████████████████████| 133 kB 15.1 MB/s
Installing collected packages: certifi, idna, urllib3, chardet, requests
  WARNING: The script chardetect is installed in '/home/damner/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed certifi-2020.6.20 chardet-3.0.4 idna-2.10 requests-2.24.0 urllib3-1.25.10
damner@codedamn:~/code$
damner@codedamn:~/code$
```

Preview of codedamn classroom

If you want to code along, you can use this free codedamn classroom that consists of multiple labs to help you learn web scraping. This will be a practical hands-on learning exercise on codedamn, similar to how you learn on freeCodeCamp.

In this classroom, you'll be using this page to test web scraping: https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/

This classroom consists of 7 labs, and you'll solve a lab in each part of this blog post. We will be using Python 3.8 + BeautifulSoup 4 for web scraping.

Part 1: Loading Web Pages with 'request'

This is the link to this lab.

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:

```python
import requests

res = requests.get('https://codedamn.com')

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

Passing requirements:

- Get the contents of the following URL using `requests` module: **https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/**
- Store the text response (as shown above) in a variable called `txt`
- Store the status code (as shown above) in a variable called `status`
- Print `txt` and `status` using `print` function

Once you understand what is happening in the code above, it is fairly simple to pass this lab. Here's the solution to this lab:

```python
import requests

# Make a request to https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/
# Store the result in 'res' variable
res = requests.get(
    'https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/')
txt = res.text
status = res.status_code

print(txt, status)
# print the result
```

Let's move on to part 2 now where you'll build more on top of your existing code.

Part 2: Extracting title with BeautifulSoup

This is the link to this lab.
In this whole classroom, you'll be using a library called `BeautifulSoup` in Python to do web scraping. 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:

```
from bs4 import BeautifulSoup


page = requests.get("https://codedamn.com")
soup = BeautifulSoup(page.content, 'html.parser')
title = soup.title.text # gets you the text of the <title>(...)</title>
```

Passing requirements:

- Use the `requests` package to get title of the URL: https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/
- Use BeautifulSoup to store the title of this page into a variable called `page_title`

Looking at the example above, you can see once we feed the `page.content` inside BeautifulSoup, you can start working with the parsed DOM tree in a very pythonic way. The solution for the lab would be:

```
import requests
from bs4 import BeautifulSoup

# Make a request to https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/
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.text

# print the result
print(page_title)
```

This was also a simple lab where we had to change the URL and print the page title. This code would pass the lab.

Part 3: Soup-ed body and head

This is the link to this lab.

In the last lab, you saw how you can extract the `title` from the page. It is equally easy to extract out certain sections too.

You also saw that you have to call `.text` on these to get the string, but you can print them without calling `.text` too, and it will give you the full markup. Try to run the example below:

```
import requests
from bs4 import BeautifulSoup

# Make a request
page = requests.get(
    "https://codedamn.com")
soup = BeautifulSoup(page.content, 'html.parser')

# Extract title of page
```

```
page_title = soup.title.text

# Extract body of page
page_body = soup.body

# Extract head of page
page_head = soup.head

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

Let's take a look at how you can extract out `body` and `head` sections from your pages.

Passing requirements:

- Repeat the experiment with URL: `https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/`
- Store page title (without calling .text) of URL in `page_title`
- 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.

The solution of this example would be simple, based on the code above:

```
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)
```

Part 4: select with BeautifulSoup

This is the link to this lab.
Now that you have explored some parts of BeautifulSoup, let's look how you can select DOM elements with BeautifulSoup methods.

Once you have the `soup` variable (like previous labs), you 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 you will select elements with CSS. Let's look at an example:

```
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 first <h1>(...)</h1> text
first_h1 = soup.select('h1')[0].text
```

`.select` returns a Python list of all the elements. This is why you selected only the first element here with the `[0]` index.

Passing requirements:

- Create a variable `all_h1_tags`. Set it to empty list.

- Use `.select` to select all the `<h1>` tags and store the text of those h1 inside `all_h1_tags` list.
- Create a variable `seventh_p_text` and store the text of the 7th `p` element (index 6) inside.
  The solution for this lab is:

```
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)
```

Let's keep going.

Part 5: Top items being scraped right now

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

If you open this page in a new tab, you'll see some top items. In this lab, your task is to scrape out their names and store them in a list

called `top_items`. You will also extract out the reviews for these items as well.
To pass this challenge, take care of the following things:

- Use `.select` to extract the titles. (Hint: one selector for product titles could be `a.title`)
- Use `.select` to extract the review count label for those product titles. (Hint: one selector for reviews could be `div.ratings`) Note: this is a complete label (i.e. **2 reviews**) and not just a number.
- Create a new dictionary in the format:

```
info = {
   "title": 'Asus AsusPro Adv...   '.strip(),
   "review": '2 reviews\n\n\n'.strip()
}
```

- Note that you are using the `strip` method to remove any extra newlines/whitespaces you might have in the output. This is **important** to pass this lab.
- Append this dictionary in a list called `top_items`
- Print this list at the end

There are quite a few tasks to be done in this challenge. Let's take a look at the solution first and understand what is happening:

```
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)
```

Note that this is only one of the solutions. You can attempt this in a different way too. In this solution:

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

Straightforward right?

Part 6: Extracting Links

This is the link to this lab.
So far you have seen how you can extract the text, or rather innerText of elements. Let's now see how you can extract attributes by extracting links from the page.

Here’s an example of how to extract out all the image information from the page:

```
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
image_data = []

# Extract and store in top_items according to instructions on the left
images = soup.select('img')
for image in images:
    src = image.get('src')
    alt = image.get('alt')
    image_data.append({"src": src, "alt": alt})

print(image_data)
```

In this lab, your task is to extract the `href` attribute of links with their `text` as well. Make sure of the following things:

- You have to create a list called `all_links`
- In this list, store all link dict information. It should be in the following format:

```
info = {
   "href": "<link here>",
   "text": "<link text here>"
}
```

- Make sure your `text` is stripped of any whitespace
- Make sure you check if your `.text` is None before you call `.strip()` on it.
- Store all these dicts in the `all_links`

- Print this list at the end

  You are extracting the attribute values just like you extract values from a dict, using the `get` function. Let's take a look at the solution for this lab:

```
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)
```

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

Part 7: Generating CSV from data

This is the link to this lab.
Finally, let's understand how you can generate CSV from a set of data. You 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`. The CSV boilerplate is given below:

```
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')

all_products = []

products = soup.select('div.thumbnail')
for product in products:
    # TODO: Work
    print("Work on product here")



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)
```

You have to extract data from the website and generate this CSV for the three products.

Passing Requirements:

- 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 your **script.py** file
  Let's see the solution to this lab:

```python
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)
```

The `for` block is the most interesting here. You extract all the elements and attributes from what you've learned so far in all the labs. When you run this code, you end up with a nice CSV file. And that's about all the basics of web scraping with BeautifulSoup!