

42 Hack High School Web-Scraping with Beautiful Soup in Python

By thomkim

What is Web-Scraping?

Web-scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. ([web_scraping](#))

Why would you want to web-scrape?

You might already be thinking numerous reasons for why to web-scrape. Here are some:

- 1) Too much information is on the internet today. You only want to know what you want to know. Get the information organized and analyzable.
- 2) A popular use is to search for online deals like airline tickets, concerts, etc. You can write a script that could scrape a website when ticket sales go online, and use bot to purchase the best tickets.
- 3) You want to analyze the Sports data, but you don't have a data to analyze. You can do this project and learn to collect data.

NOTE: I know that such power sounds too powerful for some, which is totally natural. There can be enormous benefits to web scraping, and that is very why it can also be harmful to others, if used wrong. We will go through some of the reasons and cases why you should not scrape some sites you shouldn't. Great Power comes with Great Responsibility.

We will use Python to parse (work with) HTML with help of the Beautiful Soup library. Use Python2. [Python is pre-installed in OS X. Open up Terminal and type `python --version`. You should see your python version is 2.7.x.]

By the way,

Before any of this Web-scraping, what is Web anyways?

The Web, or World Wide Web, is basically a system of internet servers that support specially formatted documents. The documents are formatted in a markup language called HTML (Hyper Text Markup Language).

Wait a second... Why are we talking about these languages like HTML? I thought we were web-scraping. Isn't it just all about the image on your screen and capturing them?

Yes, that's also another way to capture and store the data you want from the web. However, that's not technically called web-scraping, but rather image-capturing. Web-scraping does not directly deal with what's on your screen. It deals with these "specially formatted documents" that power the webpage to be rendered and posted on your screen. How web-scraping happens is looking into these documents (HTML, etc) and "extract" the necessary information.

For example,

When you visit a web page, let's say Instagram, the web browser (Chrome or Safari) makes a request to a web server (Instagram Company's Computer). Then the server sends back files to and tells your browser how to render the page. Here are the main file types that are being sent:

- 1) HTML – Contain the main content of the page
- 2) CSS – Add styling to make the page look nicer
- 3) JS – JavaScript files add interactivity to web pages
- 4) Images — image formats, such as JPG and PNG allow web pages to show pictures.

Out of all these main types, we will focus on one main type: HTML. All these files are required to make a complete page rendered, but when we are scraping, we are mainly interested on HTML.

Here are some basics about the HTML.

If you are familiar with HTML and <tags>, skip this part.

Let's create a file called “web_scraping_42.html”

```
1 <html>
2   <head>
3     <body>
4       <h1>
5         Welcome to Web Scraping by 42
6       </h1>
7       <p>
8         Welcome to Web Scraping by 42
9       </p>
10    </body>
11  </head>
12 </html>
```

Now, on your terminal, type the command “open web_scraping_42.html.”

Welcome to Web Scraping by 42

Welcome to Web Scraping by 42

Congratulations. You just created your first webpage. Very simple one though.

Every brackets `<>` are called “Tags.” Every tag starts with `<***>` then ends with `</***>`. If either of it is missing, it will error.

The most basic tag would be `<html>` tag. This tells the web browser that everything inside of it is HTML.

Then, inside of `<html>` tag, we have `<body>` tag.

Then, inside of `<body>` tag, we have `<h1>` tag and `<p>` tag. `<h1>` tag is for title heading and `<p>` tag is for paragraph. Other useful tags include `<a>` for hyperlinks, `<table>` for tables, `<tr>` for table rows, `<td>` for table columns.

As you can see, tags are nested, and can go inside other tags. You can think of it as folders inside of folders. Then, in this case, if you want to find the text that says “Welcome to Web Scraping by 42”, you start opening the folders until you reach the very last folder `<h1>` or folder `<p>`.

Also, HTML tags sometimes come with “id” or “class” attributes. The id attribute specifies a unique id for an HTML tag and the value must be unique within the HTML document. The class attribute is used to define equal styles for HTML tags with the same class. You don’t have to understand them for now. We will only make use of these ids and classes to help us locate the information we want.

Now, let's start with some Installations.

Step1:

Install necessary modules

```
$ easy_install pip
```





```
$ pip install requests
```

```
$ pip install beautifulsoup4
```

Step2:

Identify the structure of the URL and request

<https://www.49ers.com/team/players-roster/>

Player	#	Pos	HT	WT	Age	Exp	College
 Arik Armstead	91	DL	6-7	292	24	4	Oregon
 Jeremiah Attaochu	92	DL	6-3	252	25	5	Georgia Tech
 C.J. Beathard	3	QB	6-2	215	24	2	Iowa
 Ronald Blair III	98	DL	6-4	270	25	3	Appalachian State

This is the 49ers players for Rosters 2018. We can see that there is one big image on the top, then red bar, then white bar with some menus, and all these boxes below. The very bottom box is the main content that has all the statistics.

Now, let's use request and start collecting in python.

```
1 from requests import get
2
3 url = "https://www.49ers.com/team/players-roster/"
4
5 response = get(url)
6 print(response.status_code)
```

If you have successfully “get” the url, it will print out 200, which is the value of response.status_code.

```
print(response.content)
```

Then, response.content will have the HTML content of the webpage.

View easier with Graphics on Chrome with Inspect mode

Response.content is also accessible through Inspect Option on Chrome browser. Right click on Chrome webpage and click on Inspect.

The screenshot shows the San Francisco 49ers website. The top navigation bar includes links for Roster, Depth Chart, Coaches, Stats, Injury Report, and Front Office. The main content area is titled "Team Roster" and features a "Presented by" section with three buttons: "DEPTH CHART", "INJURY REPORT", and "COACHING STAFF". Below this is a table of active players.

Player	#	Pos	HT	WT	Age
Arik Armstead	91	DL	6-7	292	24
Jeremiah Attaochu	92	DL	6-3	252	25
C.J. Beathard	3	QB	6-2	215	24

The Chrome DevTools Inspect mode is open on the right side of the browser window, showing the HTML structure of the page. The "Elements" panel is active, displaying the DOM tree. The "Styles" panel is also visible, showing the default styles for the selected element.

In this mode, you can see where each line of html (on the right) is referring to on the actual page (on the left). Also, it is easier to navigate through all the <tags>. Try playing with it for a minute to get comfortable.










Now, we will use the Beautiful Soup library to parse and dive into the html to extract what we want.

Scenario:

Let's say you are a 9th grade high school football player and a 49ers Fan. You are an ambitious one as you came to 42. You want to go to the college that makes the most NFL rosters. You want to predict which college will have the most players of rosters by the time you graduate. So, you decide to collect all the necessary information...

QUEST:

Print all the names of College of 49ers Rosters 2018

 SEASON TICKETS 								
Roster Depth Chart Coaches Stats Injury Report Front Office History								
Active								
^ Player	#	Pos	HT	WT	Age	Exp	College	
 Arik Armstead	91	DL	6-7	292	24	4	Oregon	
 Jeremiah Attaochu	92	DL	6-3	252	25	5	Georgia Tech	
 C.J. Beathard	3	QB	6-2	215	24	2	Iowa	
 Ronald Blair III	98	DL	6-4	270	25	3	Appalachian State	
 Victor Bolden Jr.	17	WR	5-8	178	23	2	Oregon State	
 Kendrick Bourne	84	WR	6-1	203	22	2	Eastern Washington	
 Matt Breida	22	RB	5-10	190	23	2	Georgia Southern	

Instructions

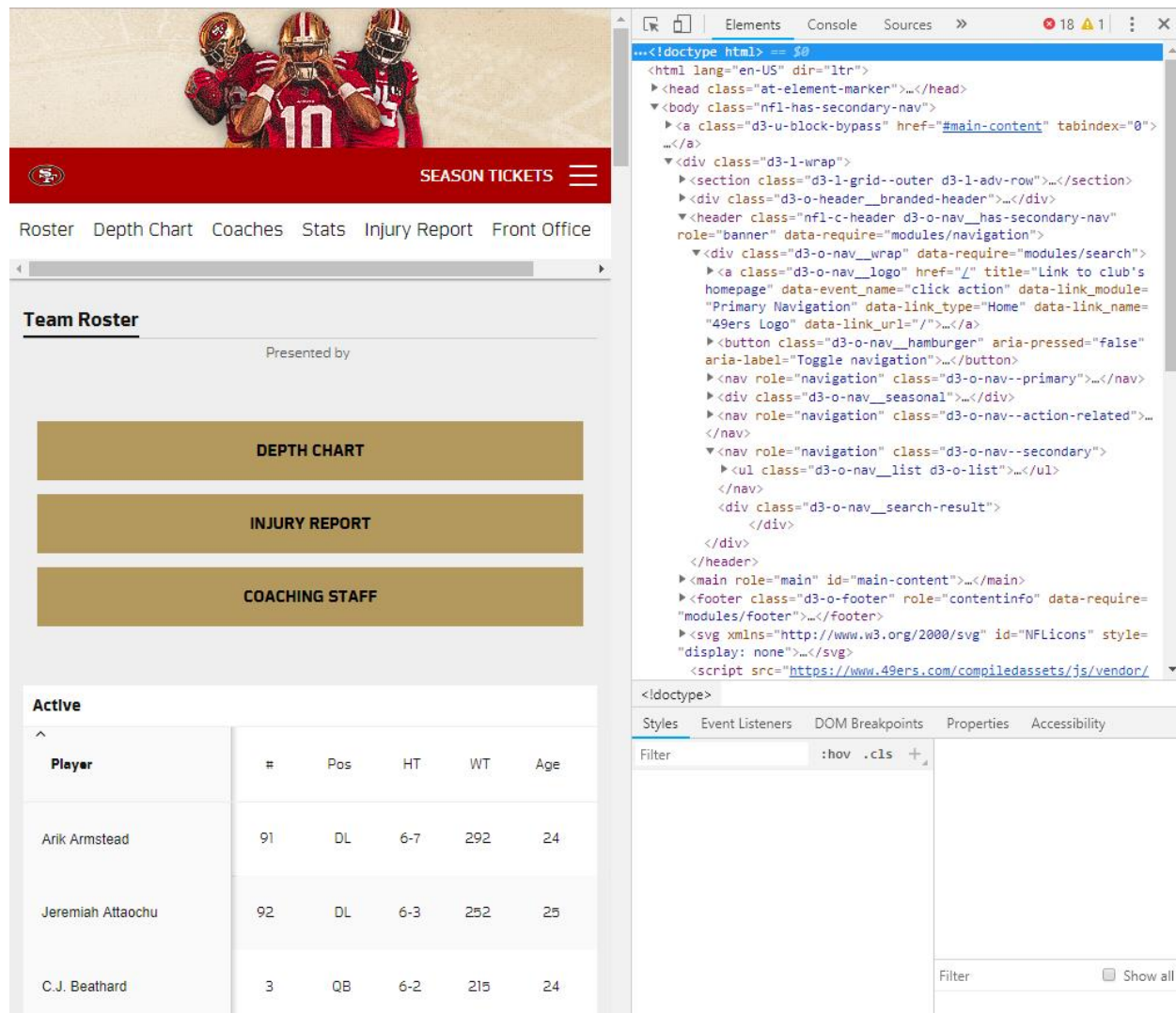
Step 1: Get URL

```
from requests import get

url = "https://www.49ers.com/team/players-roster/"

response = get(url)
```

Step 2: Open Inspect Mode for easier view on HTML with Graphics.



The screenshot shows the San Francisco 49ers website with the Chrome DevTools Inspect Mode open on the right. The website displays the team roster page, including a header with the 49ers logo and navigation links, and a table of active players.

Team Roster

Presented by

DEPTH CHART

INJURY REPORT

COACHING STAFF

Active

Player	#	Pos	HT	WT	Age
Arik Armstead	91	DL	6-7	292	24
Jeremiah Attaochu	92	DL	6-3	252	25
C.J. Beathard	3	QB	6-2	215	24

The DevTools Inspect Mode shows the HTML structure of the page, including the header, navigation menu, and the table of active players.

Step 3: Search through HTML on the Inspect Mode to find the data you want.

The screenshot shows the San Francisco 49ers website with the 'Team Roster' section highlighted. The roster table lists players with columns for Player, #, Pos, HT, WT, Age, Exp, and College. The 'Active' tab is selected, showing players like Anik Armstead, Jeremiah Attaochu, C.J. Beathard, Ronald Blair III, and Victor Bolden Jr. The Chrome DevTools Inspect Mode is open on the right, showing the HTML structure of the page. The 'Team Roster' table is highlighted in the DOM tree, and the 'Active' tab is selected in the 'Elements' panel.

Player	#	Pos	HT	WT	Age	Exp	College
Anik Armstead	91	DL	6-7	292	24	4	Oregon
Jeremiah Attaochu	92	DL	6-3	252	25	5	Georgia Tech
C.J. Beathard	3	QB	6-2	215	24	2	Iowa
Ronald Blair III	98	DL	6-4	270	25	3	Appalachian State
Victor Bolden Jr.	17	WR	5-8	178	23	2	Oregon State

Step 4: Use the location of the data and BeautifulSoup4 library to parse and extract the data

```
from bs4 import BeautifulSoup
nfl = BeautifulSoup(response.content, 'html.parser')
```

Now inside nfl, we have the content, parsed using the python built-in parser for html. We can look at it formatted even nicer with prettify() method on the BeautifulSoup object:

```
print(nfl.prettify())
```

(To Kai and others: I am not sure whether I should put “Child, Parent, and Sibling” terms here and have those features as a part for BeautifulSoup tutorial. It could be confusing. To get college name, we have a usage of “find_all” or “find” method from bs4 that are super useful. I will demonstrate with it for now, but I can also put the child parent and sibling later if needed)

Step 5: Navigate through the HTML. The line that says “<td>Oregon</td>” is where we want to get to. How to approach? Use find() or find_all().

“Oregon” may not be the same as on yours. Actually, the possibility of you having Oregon there is small. It’s because every time you request with above command, you will get a different result. For this instruction, we will use “Oregon”

```
▼<div class="d3-l-wrap">
  ▶<section class="d3-l-grid--outer d3-l-adv-row">...</section>
  ▶<div class="d3-o-header__branded-header">...</div>
  ▼<header class="nfl-c-header d3-o-nav__has-secondary-nav" role="banner" data-require=
"modules/navigation">
    ▼<div class="d3-o-nav__wrap" data-require="modules/search">
      ▶<a class="d3-o-nav__logo" href="/" title="Link to club's homepage" data-event_name=
"click action" data-link_module="Primary Navigation" data-link_type="Home" data-
link_name="49ers Logo" data-link_url="/">...</a>
      ▶<button class="d3-o-nav__hamburger" aria-pressed="false" aria-label="Toggle
navigation">...</button>
      ▶<nav role="navigation" class="d3-o-nav--primary">...</nav>
      ▶<div class="d3-o-nav__seasonal">...</div>
      ▶<nav role="navigation" class="d3-o-nav--action-related">...</nav>
      ▼<nav role="navigation" class="d3-o-nav--secondary">
        ▶<ul class="d3-o-nav__list d3-o-list">...</ul>
        </nav>
        <div class="d3-o-nav__search-result">
          </div>
      </div>
    </header>
  ▼<main role="main" id="main-content">
    ▶<section class="d3-l-grid--outer d3-l-adv-row">...</section>
    ▶<section class="d3-l-grid--outer d3-l-section-row">...</section>
    ▶<section class="d3-l-grid--outer d3-l-section-row">...</section>
    ▼<section class="d3-l-grid--outer d3-l-section-row">
      ▼<div class="d3-l-grid--inner">
        ▼<div class="d3-l-col__col-12">
          ▼<div class="nfl-o-roster">
            ▶<h4 class="nfl-o-roster__title">...</h4>
            ▼<div class="d3-o-table--horizontal-scroll">
              ▼<table summary="Roster" class="d3-o-table d3-o-table--row-striping d3-o-table-
-detailed d3-o-table--sortable {sortlist: [[0,0]]}" data-require="modules/
tableSortable">
                <caption hidden class="d3-o-table__caption">
                  Active
                </caption>
                ▶<thead>...</thead>
                ▼<tbody>
                  ▼<tr>
                    ▶<td class="sorter-lastname selected">...</td>
                    <td>91</td>
                    <td>DL</td>
                    <td class="sorter-height">6-7</td>
                    <td>292</td>
                    <td>
                      24
                      </td>
                    <td>4</td>
                    <td>Oregon</td> == $0
                  </tr>
                </tbody>
              </table>
            </div>
          </div>
        </div>
      </section>
    </main>
  </div>
```

```
nfl_div = nfl.find_all('div')
print(nfl_div)
```

The problem here is that this will print out all divs in your nfl, which is the whole script. So, that's why you are seeing all of that.

So, let's analyze the html to figure out how we will use find() method.

```
<!doctype html>
<html lang="en-US" dir="ltr">
  <head class="at-element-marker">...</head>
  <body class="nfl-has-secondary-nav">
    <a class="d3-u-block-bypass" href="#main-content" tabindex="0">...</a>
    <div class="d3-l-wrap">
      <section class="d3-l-grid--outer d3-l-adv-row">...</section>
      <div class="d3-o-header__branded-header">...</div>
      <header class="nfl-c-header d3-o-nav__has-secondary-nav" role="banner" data-require="modules/
navigation">...</header>
      <main role="main" id="main-content">
        <section class="d3-l-grid--outer d3-l-adv-row">...</section>
        <section class="d3-l-grid--outer d3-l-section-row">...</section>
        <section class="d3-l-grid--outer d3-l-section-row">...</section>
        <section class="d3-l-grid--outer d3-l-section-row">
          <div class="d3-l-grid--inner">
            <div class="d3-l-col col-12">
              <div class="nfl-o-roster">
                <h4 class="nfl-o-roster__title">...</h4>
                <div class="d3-o-table--horizontal-scroll">
                  <table summary="Roster" class="d3-o-table d3-o-table--row-striping d3-o-table--
detailed d3-o-table--sortable {sortlist: [[0,0]]}" data-require="modules/
tableSortable">
                    <caption hidden class="d3-o-table__caption">
                        Active
                    </caption>
                    <thead>...</thead>
                    <tbody>
                      <tr>
                        <td class="sorter-lastname selected">...</td>
                        <td>91</td>
                        <td>DL</td>
                        <td class="sorter-height">6-7</td>
                        <td>292</td>
                        <td>
                            24
                        </td>
                        <td>4</td>
                        <td>Oregon</td> == $0
                      </tr>
                      <tr>...</tr>
                      <tr>...</tr>
                      <tr>...</tr>
                      <tr>...</tr>
                    </tbody>
                  </table>
                </div>
              </div>
            </div>
          </section>
        </main>
      </div>
    </body>
  </html>
```

The underlined with red are basically the parent (above folders) tags of the data we are looking for, which is in this case “Oregon”.

Let’s parse from nfl (the whole script) to <main> tag with id=’main-content’. Remember we talked about how we can use the attributes such as id in web-scraping?

```
nfl_main = nfl.find(id="main-content")
print(nfl_main.prettify())
```

Now, you see everything below <main id="main-content" role="main">

It is your turn to try now. Play around and explore the usage of BeautifulSoup4 methods. You only saw one method, which is find(). man BeautifulSoup4

Try Yourself

Try capturing the “Oregon” (or whatever is your “FIRST” player’s College name) and print it out on your terminal.

Bonus #1

Print out all the players’ College names on your terminal.

If you have done both questions above, then now you are ready to store (save) the data and neatly make it into CSV format, which is widely used format for analysis. Here is how to put “Try Yourself” College answer into your CSV format.

```
#First, Import your Python csv module
import csv
# open a csv file and write your "first_name" on to it and
# export it to csv file called first_name.csv
with open('first_name.csv') as csv_file:
    writer = csv.writer(csv_file)
    writer.writerow([first_name])
```

Now, if you look at your current directory, it will have a file called ‘first_name.csv’.

Check it out yourself. Now it’s your turn to try.

Try Yourself

Store all the players' College names into a csv file called "college.csv"

Bonus #2

Store all the players' whole data (Name, Number, Position, Height, Weight, Age, Experience, and College) into a csv file called "roster_table.csv"

Scraping Rules

Remember that we will deal with some rules regarding whether it is legal or illegal for some sites for scraping. Here are three from Google.

- You should check a website's Terms and Conditions before you scrape it. Be careful to read the statements about legal use of data. Usually, the data you scrape should not be used for commercial purposes.
- Do not request data from the website too aggressively with your program (also known as spamming), as this may break the website. Make sure your program behaves in a reasonable manner (i.e. acts like a human). One request for one webpage per second is good practice.
- The layout of a website may change from time to time, so make sure to revisit the site and rewrite your code as needed

Make sure you are not violating the terms!