

Web Scraping for Dummies

Introduction

Here, I discuss the topic which has received the limelight recently and has become an essential area of expertise for the projects/companies that require huge amount of data. In recent years, the most popular and widespread systems being used are empowered by **machine-learning, analytics and, big data**. The common requirement among all these systems **is the use of big datasets**.


Web scraping V/S API call

Data extracted through API calls have a lot of limitations as they are huge in number and go through the strenuous process of downloading and curating. One of the main reasons is that no one really cares about or maintains structured data. If it goes offline or gets horribly mingled, no one really notices. Conversely, in case of Web Scraping you don't need to wait for a site to open up an API or even contact the website owner. Just invest some time browsing the site until you find the data you need and figure out some basic access patterns. This reduces the curation efforts to less than 50%.


Basics of web scraping

Before scraping a website it's very important to diagnose and analyse the website, before trying to fetch data from it. Let's have a look at the basic key aspects to consider in order to analyse a website for scraping:

1. You'll need to start by finding your “**endpoints**” — the URL or URL that return the data you need.
2. There's usually some sort of pagination issue which does not allow you to see all of it at once. Usually, clicking to next page adds some sort of “*page=<params>*” or “*offset=<params>*” parameters to the URL, which is usually either the page number or else the number of items displayed on the page.




Now, you've figured out the pattern of the URL of the data you need from the server. Here comes, the somewhat tricky part; getting the data you need out of the page's markup. You can jump to the parts of the markup that contain the data you need.



Just right click on a section of information you need and pull up the **Web Inspector** to look at it. Zoom up and down through the **DOM** tree until you find the outermost **<div>** around the item you want. Once you find the right node in the **DOM** tree, you should always view the source of the page (“**right click**” > “**View Source**”) to make sure the elements you need are actually showing up in the raw **HTML**.

It is probably not a good idea to try parsing the HTML of the page as a long string. Spend some time doing research for a good HTML parsing library in your language of choice. A good library will read the HTML that you pull in using some HTTP library and turn it into an object that you can traverse and iterate over to your heart's content.



Beautiful Soup is a Python library for pulling data out of HTML and XML files. It provides idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves hours or days of your work.

The examples in here should work the same way in Python 2.7 and Python 3.2.

Installing BeautifulSoup

If you're using a recent version of Ubuntu Linux, you can install **Beautiful Soup** with the system package manager:

```
$ apt-get install python-bs4
```

```
$ easy_install beautifulsoup4
```

```
$ pip install beautifulsoup4
```