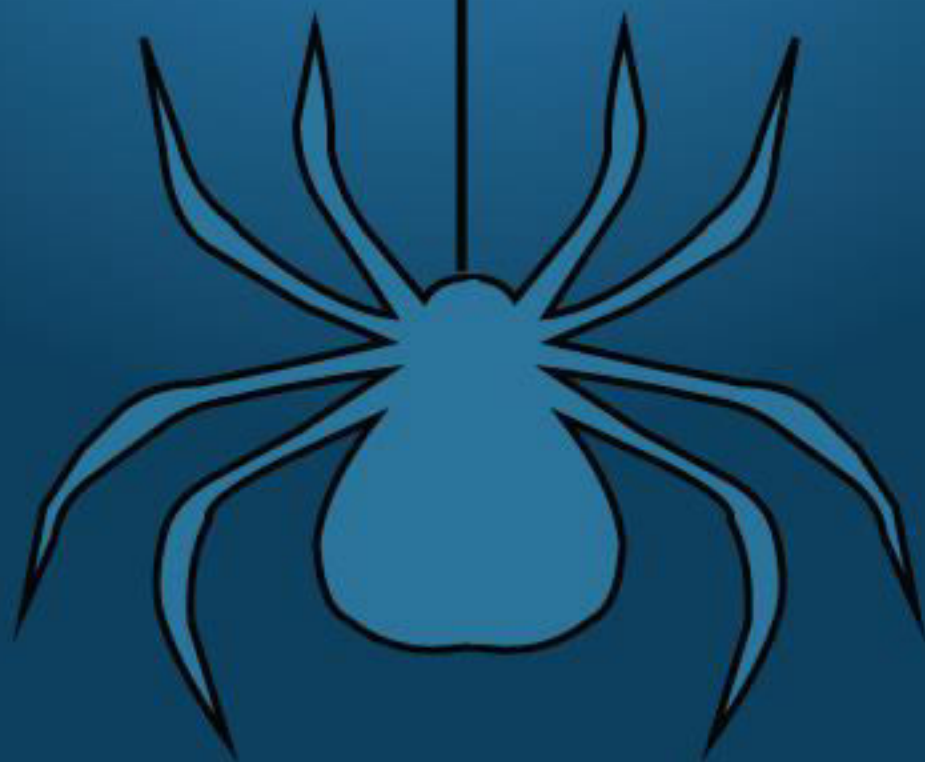


**Learn
Web
Scraping**

**From
Scratch**



Learn Web Scraping From Scratch

by Attila Toth

Copyright © 2016 Attila Toth

<http://ScrapingAuthority.com>

Table of Contents

Introduction.....4

Web Scraping in General.....10

5-Step Data Collecting Process.....14

Selectors in Web Scraping.....17

BeautifulSoup.....24

Scrapy Framework.....30

Ethical Web Scraping.....46

Where to Go Next.....51

Introduction



Hey, I'm Attila Toth.

Before we jump into learning about web scraping, crawling techniques and best practices to advance your web scraping skills, I wanted to tell you a bit about myself and why this book was created.

So let's kick it off with a few things about me:

- **I'm a student studying programming.**
- **I read a lot, about 40-50 nonfiction books a year.**
- **I like to share my knowledge.**

Of course, having a blog and ebook about web scraping, you can tell this is a thing I love to do as well. Now I'm going to share with you the little story how it all started and why I started to learn web scraping.

Some years ago I was obsessed with sport betting. I didn't do it for the money I just enjoyed the excitement in it. I placed bets inconsistently for about half a year. Then I realized it was not fun at all. Not that I lost so much amount of money (I played safe) I was missing the joy of winning. At the time I was already a decent programmer so I figured out I could create a software that would predict sport event outcomes. My only job would be to place my bets according to the software.

There was an algorithm in my mind which would analyze sport statistics and show me the "value bets". So there I was with a nice plan on how to win most of my bets I would place. Though I've got a problem. I didn't have any source of sport database or such. Worse, I could not find anything on the internet which was appropriate for me.

It was some weeks or months later when I realized there were some websites that had the information I needed. The problem was that the data on these websites was unstructured and not gathered in any kind of database.

At the time I had no idea about keywords like “web scraping” or “web crawling” so I just searched for “download data from website” or something like that in google. As I came across web scraping and crawling, I started to learn about html parsing libraries and Scrapy. Later I reached a point when I could create my own web scraper. Fortunately, I found a website with the data I needed for my software, with a really old-fashioned and unmaintained layout so it was easy to scrape from.

I was able to fetch unstructured data and transform it into a database. Though, with the database in my hand, I was able to code my betting software, my data analyzing and betting strategy didn't work out as I expected.

So that was my first experience with web scraping. It was fun and I learnt a lot. Later I recognized how companies use web scraping technologies without hurting or harming other parties yet benefiting themselves.

Since then, I've created hundreds of scrapers and crawlers. I'm supposed to mention that ethical and lawful web scraping is the only way to do it. Without hurting or stealing anything, you can (and should) read more about ethical web scraping at the end of this book.

About the book

This book was created to help you start out with web scraping. First, I will give you a quick introduction to web scraping in general and what to do before diving into coding. The next chapter is going to tell you how to write efficient selectors and xpath to parse html, which you always need to do while scraping.

Later, we will dive into coding. You will learn the basics of html parsing and how to scrape the web with BeautifulSoup. Then, we will compare BeautifulSoup with a crawling framework, Scrapy. In that chapter you will create your first Scrapy crawler and have an idea how Scrapy works. Also, you'll be able to extract information in any format using Item Pipelines.

At the end of the book, there's a chapter on how to create ethical and lawful web crawlers which is essential to not get in trouble while crawling the web.

Who is it for

If this book had been written when I searched for web scraping learning material for beginners I would have picked it up immediately. This book is intended to help you start web scraping. You don't need any knowledge or experience using any web scraping library or such. So if you are a novice web scraper this is the best book for you! If you have some experience with web crawling you might find some of the chapters boring. For you, I suggest sticking around my [Twitter](#) and signup to my email list on [Scraping Authority blog](#) to notice when I will publish my web crawling course for advanced fellows. (I'm working on it now)

For those, who want the best available learning material right now, I highly recommend my friend [Hartley Brody's web scraping course](#).

How to reach out to me

I'm always glad to help whenever I can. Feel free to reach out to me if you have a question or something to say. Also, if you think that some of the chapters don't make sense, let me know!

- **Email:** attila@scrapingauthority.com
- **Twitter:** <http://twitter.com/scrapingA>
- **Facebook:** <https://www.facebook.com/scrapingauthority/>
- **Website:** <http://www.scrapingauthority.com/>

If you haven't already signed up for my Web Scraping email list and you're interested: <http://www.scrapingauthority.com/signup>

I really appreciate you picked up my book, I hope you will like it!

ATTILA TOTH

Web Scraping in General



When I tell people I'm *passionate* about web scraping they often misinterpret it this way:

"I don't really understand why anyone could become passionate about doing something that essentially takes the work of other people and then repurposes their data for a profit without permission."

Today I came up with a metaphor to illustrate what web scraping is really about:

Imagine a cute puppy in front of you. You try to pet her and play with her because it's a cute puppy. You try to touch her but you feel something cold that hits your hand but you cannot see anything. There's a thick glass between the puppy and you. You just stand there and look at the puppy and miss the opportunity to have fun.

Web scraping gives you the option to bypass the glass. Also, you have to be careful and find a way not to break the glass!

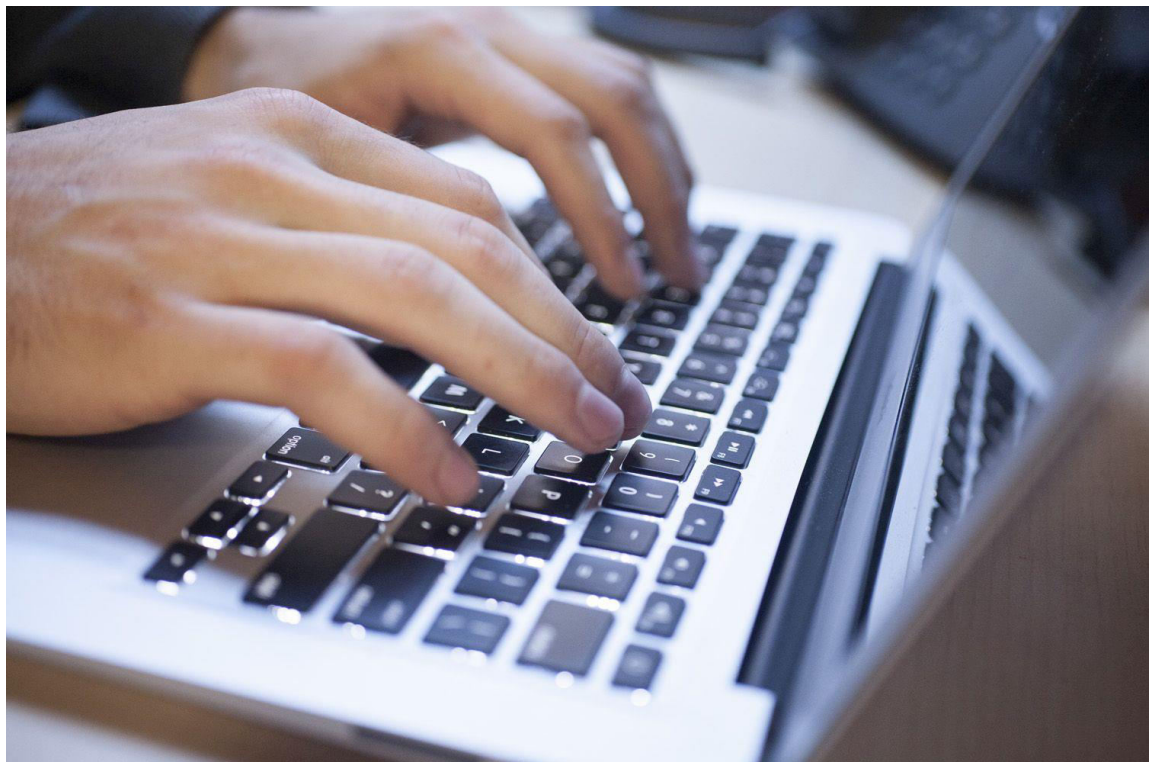
What is Web Scraping

Web Scraping is the process when we transform unstructured data and information, fetched from a certain website by our web scraper, to a structured database.

I use this interpretation because the main goal of web scraping is to make data useful for people. Without transforming unstructured information into structured one so you can actually use it, web scraping is pointless.

In technical aspects, web scraping is the art of collecting data out of an html file by parsing then processing it into database.

What I really love about web scraping is the idea to make use of the huge amount of unstructured data on the internet. Also, as an advocator of *Open Data*, web scraping plays a big role in making it happen.



How is it Beneficial

How is web scraping beneficial?

As I mentioned earlier the real beauty in web scraping is that you can make use of data which you couldn't otherwise. Nowadays there are so much information on the web that I guarantee you there is no business which couldn't benefit from data collection from the web. That is why about 80% of online businesses have a sort of web crawling system established that enables them to harvest data from the web.

For example, a friend of mine has a business in the hospitality industry. In order to keep up the pace, he has to use web scrapers to monitor competitors' websites regularly. If he didn't do it chances are he would be out of business very soon.

To give more ideas what you can create using web scraping I came up with a list:

- Discover product defects by scraping product reviews on message boards and forums.
- Extract product images and specification documents
- Extract bond and stock pricing history
- Extract job listings
- Monitor competitor's inventory information
- Check the meta information of websites
- Etc... There's a bunch of other things you can do

The 5-Step Online Data Collecting Process

So here we are, close to jumping into the coding part of this book and create our first scraper but now you should learn the important steps to take before building your web scraper.

Here are the five steps to gather information online:

1. **Recognize what data you really need**
2. **Find source(s) where you have access to it**
3. **Use API if you can**
4. **Build your scraper**
5. **Process data**

This is the system you should use in every iteration when you are about to gather data from websites.

The sole goal of this process is to make sure that you don't waste any time. If you don't identify very first what data you (or your company) need exactly in what format you risk that you will have to do the work again. Or if you build a scraper to fetch data from a website which has a pretty useful API you waste your time.

So now let's breakdown the steps!

1. Recognize what data you really need

In the first step you should gain a clear understanding of what type of data you really need and in what format (JSON, SQL, etc...).

2. Find source(s) where you can access it

When you clearly know what you need you can search for websites that have it. Try to find an online source where you can immediately download the data you need in a database. If you are lucky, you are able to get your job done in this step. In most cases, you won't be so lucky to get a quick out-of-the-box database so you should keep looking for an appropriate website. If you are after a specific data which is not website-dependent you should choose a website which has a simple layout and the least complicated html structure. If you want to collect information from a specific website this step is done immediately.

3. Use API if you can

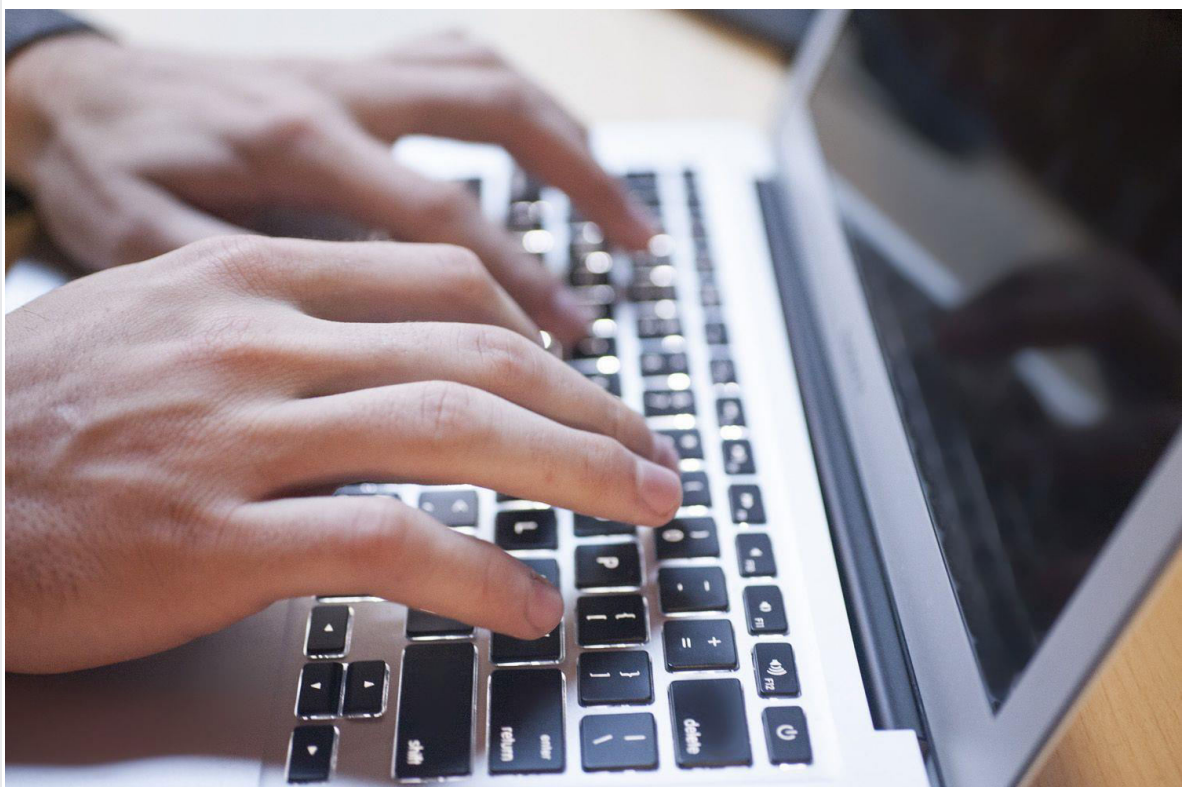
Good news is that nowadays more and more website has its own API for developers which provides access to their information. In this step you should look for some kind of API of the website. Keep in mind, the website doesn't have to indicate they have a working API so when you are about to scrape, give it a try and ask the administrator of the website if there is accessible API for developers. If so, you should ascertain whether it is an API which cover all the data the website has or just a portion of it to figure out if it fits your needs.

4. Build your scraper

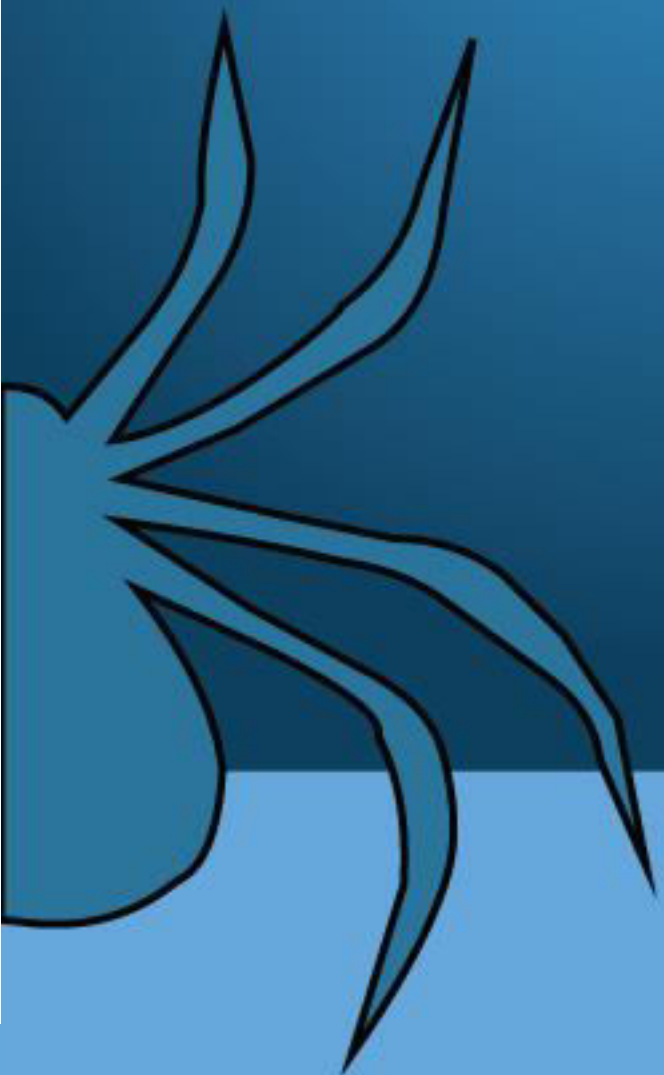
No existing database, no API. If you are at this point, you've got nothing else to do but actually create your scraper. You will learn how to do it properly in the next chapters.

5. Process data

Processing data is the last step to collect data from the web. In this phase, you transform the plain information into a useful database. The database can be a simple JSON or CSV, or a more complex DB like SQL or whatever you desire. If you're done with this, the data is usable and interpretable by a human or a software as well.



Selectors in Web Scraping



Selectors in Web Scraping

Selectors are one of the most important pieces of your scraper. Well-written selectors make your web scraper work efficiently and fast. When the website's layout changes your scraper's selectors need to be changed as well. Then, in a well-established scraping environment the only things that have to be changed are the selectors. In this chapter I will dig a bit into CSS selectors and XPATH and share some good tips with you on how to write effective and fast selectors for your web scraper.

**CSS
Selectors**

<XPATH>

CSS Selectors

CSS selectors are widely used by frontend developers to associate CSS properties with their html elements. For web scrapers, we can use it to navigate in the structure of an html file. If you are a beginner scraper and you're familiar with CSS then I suggest that you should use CSS selectors over XPATH, though in some cases you have to use XPATH to select the elements you want.

XPATH

XPATH is a specification which is created to help you navigate in any XML document so you can use it while you're parsing an html file. Almost each html parsing or web scraping library has XPATH support. It's a more robust and powerful way to locate elements than CSS selectors.

Basic CSS Selectors

#x

Element that has x id.

#id

.x

Elements that has x class. Selects all elements that have x class

.content

x y

Element is a direct or non-direct descendant of x.

body p

x, y

Elements that are x or y.

div, p

x + y

First element that is immediately preceded by x

div + p

x > y

Element is a direct child of x.

div > p

x ~ y

All elements that is preceded by x

p ~ ul

x[y]

Element that has y attribute.

div[alt]

x[y='z']

Element's y attribute is "z".

img[alt='image']

x:last-child

Elements that is the last child of its parent.

p:last-child

x:empty

Elements that have no children.

p:empty

Basic XPATH Expressions

/

Start searching from root node.

//

Start searching from the start of the document.

//x[@id='y']

Element that has y id.

//div[@id='foo']

//x[@class='y']

Elements that has y class.

//div[@class='foo']

//x | //y

Selects elements that are x or y. Searching in the whole document.

//H1 | //H2

//x[@y='z']

Elements that has y attribute which are z.

//img[@alt='image']

//x/y/z

Element is direct descendant of y and y is direct descendant of x.

//p/ul/li

//x/text()

Selects the text in x.

//p/text()

//x/y[N]

The Nth y element that is a child of x.

//div[@id='foo']/td[1]

You can check out the [CSS Selector Reference](#) and [Xpath Syntax](#) to learn more about selecting elements in an html document.

4 Tips to Write Effective Selectors

- Be specific if necessary and at the same time use as short selectors as you can.
- Know the HTML structure of the website thoroughly. Take time to go over it.
- Maintain the selectors. If the layout changed you probably need to change your code.
- Write selectors for yourself. Try to avoid tools.

XPATH and CSS Selector Generator Tools

It can take a lot of time to figure out and test your selectors especially if it is a large project. If you are not afraid of messy CSS Selectors or XPATH or simply you don't want to waste time writing your own selectors you can use one of the amazing tools below to make your job easier. These tools will generate your desired selectors and XPATH. Be aware that these tools don't necessarily create the most readable and most efficient piece of code. Also, they sometimes generate wrong strings that doesn't select what you need!

CSS Selector Tools

<http://selectorgadget.com/>

<https://chrome.google.com/webstore/detail/css-selector-helper-for-c/gddgceinofapfodcekopkjjelkbjodin>

<http://getfirebug.com/>

XPATH Tools

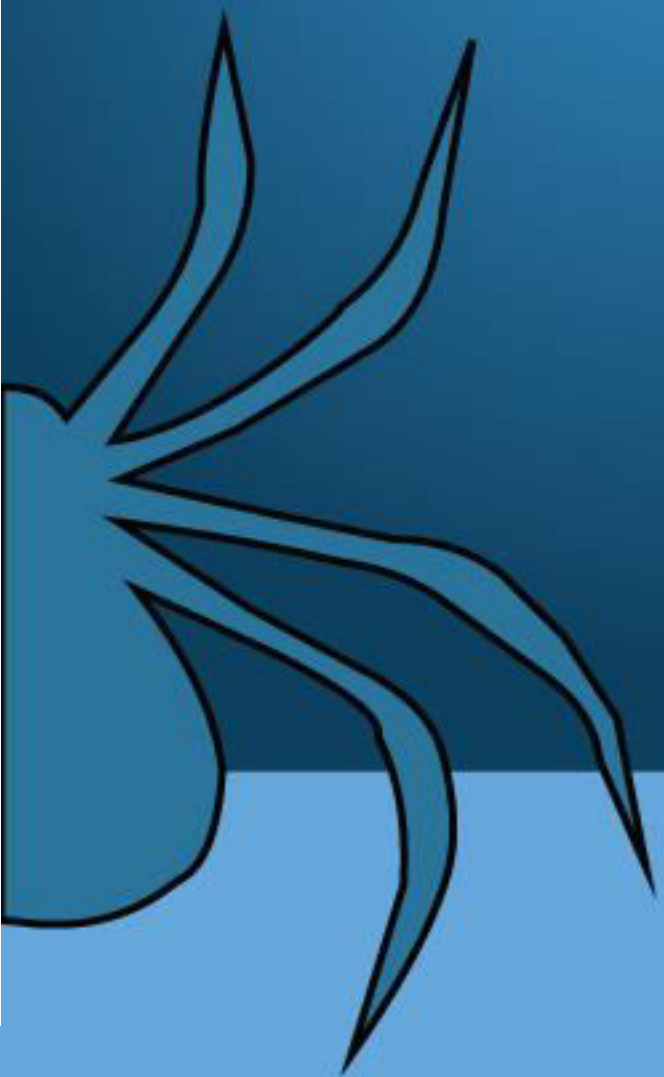
<http://www.bubasoft.net/product/xpath-builder/>

http://www.altova.com/xmlspy/xpath-analyzer.html#xpath_analyzer20

http://xmltoolbox.appspot.com/xpath_generator.html

<http://getfirebug.com/>

BeautifulSoup



Building our first Scraper with BeautifulSoup

BeautifulSoup is the most popular web scraping library in Python. It's very easy to use and you will be able to create your web scraper fast.

Install BeautifulSoup

You can install BeautifulSoup on either Linux or Windows. The most recent version of this library (beautifulsoup4) supports python 2 and python 3 too. You should already have Python installed.

Installation on Ubuntu

If you haven't already installed pip you should do it now:

```
apt-get update  
apt-get -y install python-pip
```

Then install BeautifulSoup:

```
pip install beautifulsoup4
```

Installation on Windows with PIP

If you're planning to work with python on your Windows in the future I suggest that you should install pip so you can download python packages faster and easier from the command line. You will probably need to open Command Prompt as an administrator.

Download get-pip.py from [here](#).

Open CMD and cd to the folder you've just downloaded it and run it:

```
cd C:\Users\Attila\Desktop\Folder  
python get-pip.py
```

Pip is installed on your system. You can test it by checking its version:

```
pip -V
```

Install BeautifulSoup with pip:

```
pip install beautifulsoup4
```

Web Scraping with BeautifulSoup

In this example and in the next tutorials, we are going to scrape <https://scrapethissite.com> which is a website dedicated to help you learn web scraping. Also, you can purchase my friend [Hartley Brody's web scraping](#) course there, I highly recommend it.

Because BeautifulSoup cannot load any html page from the internet you need to use a library such as [urllib2](#).

First, you need to request the page (<https://scrapethissite.com/pages/simple/>) you want to scrape then setup a proper user agent to identify yourself.

```
#create request and set user agent
request = urllib2.Request('https://scrapethissite.com/pages/simple/')
request.add_header('User-Agent', 'Scraping Authority
(ScrapingAuthority.com'))
```

```
#open page
open = urllib2.urlopen(request)
```

Now we have one more task before scraping: Determine which parser lib BeautifulSoup should use. Python has a built-on parser lib `html.parser`. In this example I'm using this one but you can choose another third party lib for example `lxml`.

```
page = BeautifulSoup(open, 'html.parser')
```

```
title = page.title.text #title of page
```

We've just setup BeautifulSoup correctly we can use our "page" object to navigate and find elements on the page.

Selectors

Using our "page" object you can navigate to any tags on the page and extract it right away:

```
h3 = page.h3.get_text() #selects the first H3 tag on the page and  
extracts its text
```

Also, you can find elements by passing the tag and class name as arguments like this:

```
country_name_tags = page.find_all('h3', class_ = 'country-name')
```

This statement selects all element which has "h3" tag and its class is "country-name".

BeautifulSoup supports CSS selectors that make you select elements very easy. Let's see an example:

```
country_name_tags = page.select('.row .col-md-4.country .country-  
name') #select country name elements
```

```
country_names = []
```

```
for country_name_tag in country_name_tags : #extract text only  
    country_names.append(country_name_tag.get_text())
```

BeautifulSoup doesn't support XPath. Check out [Darian Moody's article](#) why CSS selectors are better than XPath!

Pagination

This is an example of how to find the URLs on this page

(<https://scrapethissite.com/pages/forms/>).

```
link_tags = page.select('.pagination a')
```

```
for link_tag in link_tags:
```

```
    url = link_tag.get('href')
```

```
    # scrape url...
```

Scrapy ***Framework***



Differences between Scrapy and BeautifulSoup

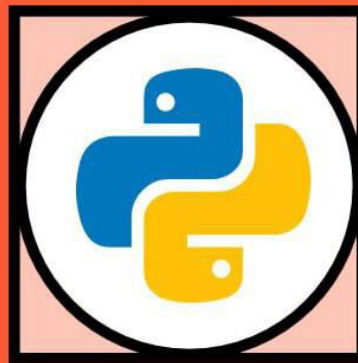
Scrapy vs. BeautifulSoup

ScrapingAuthority.com

WHAT IS IT?

Scrapy

Python framework for crawling web sites and extracting and processing structured data.



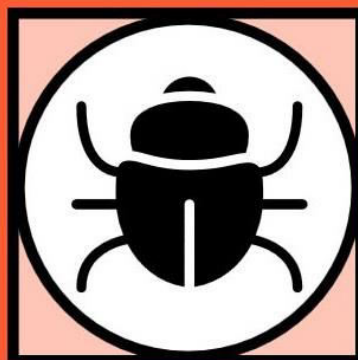
BeautifulSoup

Python library for html/xml parsing. Suitable for scraping html documents.

WHAT IS IT CAPABLE OF?

Scrapy

- download web page
- scrape information
- clean data
- save to database



BeautifulSoup

- parse html
- scrape information

WHEN TO USE IT?

Scrapy

- need to login to scrape data
- need to crawl
- in a long-term project
- need to scrape from numerous web pages
- need to process or modify data



BeautifulSoup

- no need to login to scrape data
- no need to crawl
- in a one-off project
- need to scrape from a single web page
- no need to process or modify data

DOES IT SUPPORT JS?

Scrapy

No JS support



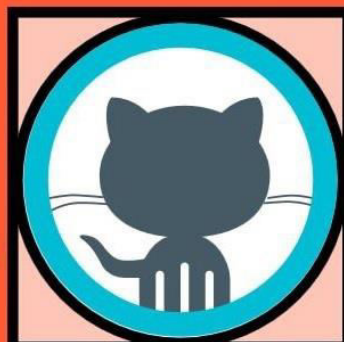
BeautifulSoup

No JS support

ADDITIONALLY?

Scrapy

- open source
- 200+ contributors on Github
- can use BeautifulSoup inside Scrapy
- supports Python 3



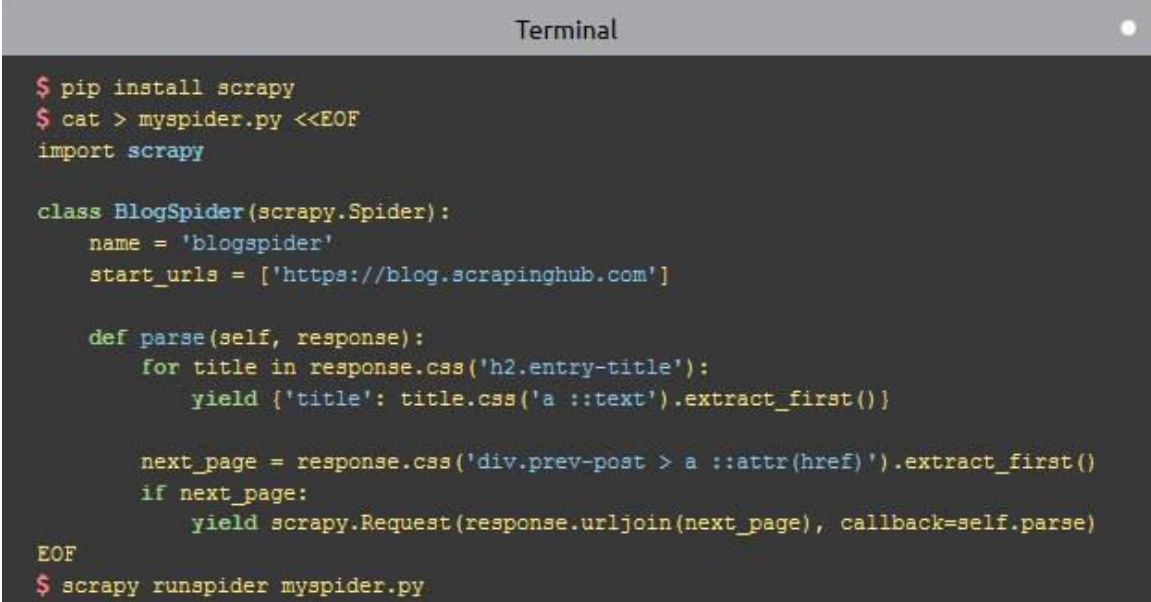
BeautifulSoup

- open source
- Reddit uses it
- supports Python 3

Why You Should Use Scrapy

Scrapy is a fast and powerful open source Web Scraping framework. If you need to have a scraping environment for long term Scrapy is the best choice because it's easily extensible and you can plug new functionality without having to touch the core. Scrapy is written in Python and runs on Linux, Windows, on local machine, on server or wherever else you desire.

A sample piece of scrapy code from scrapy.org:

A terminal window with a dark background and light-colored text. The title bar at the top says "Terminal". The content shows a series of commands and code being entered into a shell. The commands use a red prompt character '\$'. The code is a Python class definition for a Scrapy spider, using blue and green syntax highlighting. The code defines a 'BlogSpider' class that inherits from 'scrapy.Spider'. It sets a name, start_urls, and a parse method that extracts titles and follows a next page link. The terminal ends with a command to run the spider.

```
$ pip install scrapy
$ cat > myspider.py <<EOF
import scrapy

class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://blog.scrapinghub.com']

    def parse(self, response):
        for title in response.css('h2.entry-title'):
            yield {'title': title.css('a ::text').extract_first()}

        next_page = response.css('div.prev-post > a ::attr(href)').extract_first()
        if next_page:
            yield scrapy.Request(response.urljoin(next_page), callback=self.parse)
EOF
$ scrapy runspider myspider.py
```

Install Scrapy On Ubuntu

The very first step is to make sure that each dependency of Scrapy is installed locally in your system. If you followed the BeautifulSoup installation guide you have pip installed on your system already.

Scrapy dependencies

You will need these packages installed already:

- python-dev
- python-pip
- libxml2-dev
- libxslt1-dev
- zlib1g-dev
- libffi-dev
- libssl-dev

If you're not sure these packages are already installed or you want to make sure they are, just open a terminal with Ctrl+alt and run this command:

```
sudo apt-get install python-dev python-pip libxml2-dev libxslt1-dev  
zlib1g-dev libffi-dev libssl-dev
```

This command will install every dependency you will need.

Install Scrapy with pip

Now you can easily install scrapy with pip, run this command:

```
pip install Scrapy
```

You can test if Scrapy installed properly by running this:

```
scrapy version
```

Now you should see which Scrapy version is installed now.

If you have problems installing scrapy follow this link:

www.scrapingauthority.com/2016/08/06/install-scrapy-ubuntu

Scrapy Item

We use web scraping to turn unstructured data into highly structured data.

Essentially, it's the goal of web scraping. Structured data means collected information in database such as mongoDB or SQL database. Also, in most cases we only need some simple data structure such as JSON, CSV or XML. With Scrapy Items you can process data to fit your desires. In Scrapy you can use the Item class to store data before extracting into any kind of structured data system mentioned before.

To extract data with Scrapy we use particular Item classes in our project to store information we need to fetch. You can have as many items as you wish. Strive to organize your scrapy items in the most readable and logical way. Each item class has to derive from scrapy.Item class.

```
class BookExampleItem(scrapy.Item):  
    title = scrapy.Field()  
    author = scrapy.Field()  
    length = scrapy.Field()  
    paperback = scrapy.Field()  
    publisher = scrapy.Field()
```

It's as simple as that. You create Scrapy Fields in your item to store data.

Associate Fields with data

Now you know how to create Item classes. Setting values to them works the same way as you set values in a [Dictionary](#).

```
book_item = BookExampleItem()  
book_item["title"] = 'title'  
book_item["author"] = 'author'  
book_item["length"] = 'length'  
book_item["paperback"] = 'paperback'  
book_item["publisher"] = 'publisher'
```

This way you can easily set values to your Item Fields in your Spider script just like a dict.

Now you have a basic understanding of how Scrapy Items work so we can start scraping.

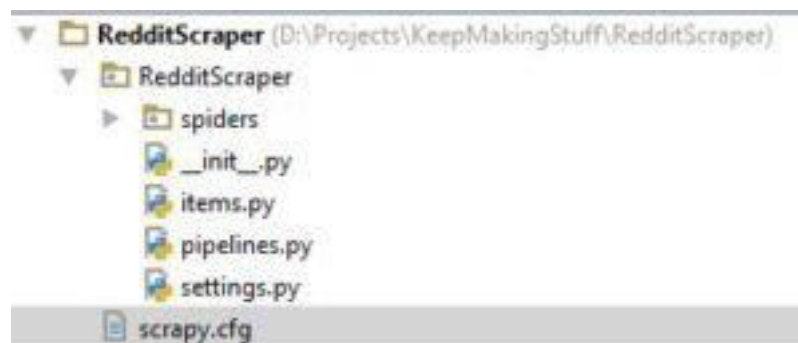
How to Scrape Single Page

We're gonna create a scraper that crawl some data from reddit's [programming subreddit](#). More specifically, we are going to crawl the hottest programming topics from the first page, their links and their posting time.

Very first, we create a scrapy project using this command:

```
scrapy startproject RedditScraper
```

Our project structure looks like this:



In Scrapy, we have to store scraped data in Item classes, as we learnt. In our case, an Item will have fields like title, link and posting_time.

```
class RedditscraperItem(scrapy.Item):  
    title = scrapy.Field()  
    link = scrapy.Field()  
    posting_time = scrapy.Field()
```

Next, we create a new python file in Spiders folder in which we're going to implement parse method. But first, we should check out the [page's](#) HTML to figure out a way to achieve the information we need.

```
<span class="rank">1</span>
▶ <div class="midcol unvoted">...</div>
▼ <div class="entry unvoted">
  ▼ <p class="title">
    <a class="title may-blank " href="http://flif.info/index.html" tabindex="1">FLIF - Free Lossless Ima
    ▶ <span class="domain">...</span>
    </p>
  ▶ <p class="tagline">...</p>
  ▶ <ul class="flat-list buttons">...</ul>
  <div class="reportform report-t3_49bs2c"></div>
  ▶ <div class="expando expando-uninitialized" style="display: none">...</div>
</div>
<div class="child"></div>
<div class="clearleft"></div>
```

We can see that each data we need is inside a class named “entry unvoted”.

Now in our custom spider we need to define its name as a string and its urls as a list. Then in parse method we create a for loop which go through each “entry unvoted” div and find every title, link and posting time. We are using xpath because now I find it more efficient and readable than CSS selectors but you can use CSS if you want.

```
import scrapy
from RedditScraper.items import RedditItem

class RedditSpider(scrapy.Spider):
    name = "reddit"
    start_urls = [ "https://www.reddit.com/r/programming" ]
    def parse(self, response):
        for selector in response.xpath("//div[@class='entry unvoted']"):
            item = RedditItem()
            item["title"] = selector.xpath("p[@class='title']/a/text()").extract()
            item["link"] = selector.xpath("p[@class='title']/a/@href").extract()
            item["posting_time"] =
            selector.xpath("p[@class='tagline']/time/text()").extract()
            yield item
```

Running the `scrapy crawl reddit` command we get what we wanted wrapped in Items:

```
2016-03-07 18:49:52 [scrapy] DEBUG: Scraped from 200
https://www.reddit.com/r/programming
{'link': [u'http://thecodelesscode.com/case/225'],
'posting_time': [u'3 hours ago'],
'title': [u'[Codeless Code] Case 225: The Three Most Terrifying Words']}
2016-03-07 18:49:52 [scrapy] DEBUG: Scraped from 200
https://www.reddit.com/r/programming
{'link': [u'https://textplain.wordpress.com/2016/03/06/using-https-properly/'],
'posting_time': [u'2 hours ago'],
'title': [u'Using HTTPS Properly']}
...
```

Pagination

Going further with web scraping, you will need to visit a bunch of URLs within a website and execute the same scraping script again and again. In my and BeautifulSoup tutorial I showed you how you can paginate on a website now you will learn how to do this with Scrapy. The [CrawlSpider module](#) makes it super easy.

As an example, we are going to scrape some data from [Amazon](#). As usual, scrapy will do most of the work and now we're using its CrawlSpider Module. It provides an attribute called [rule](#). This is a tuple in which we define rules about links we want our crawler to follow.

First and foremost, we should setup a User Agent because we want our crawler to see the site layout like we see it in browser and it's fine because we don't intend to do anything harmful. You can setup a User Agent in settings.py:

```
USER_AGENT='Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.36 Safari/535.7'
```

Scrapy CrawlSpider Rule Attribute

So for example we're looking for the most reviewed books in each category. In our spider file we create a *Rule* which contains where book category links are on the page then callback the method we want to execute inside each category page(our starting url is [amazon books page](#)):

```
rules = ( Rule(
    LinkExtractor(restrict_xpaths =
        "(//div[@class='categoryRefinementsSection']/ul/li)"),callback="sort_books"),)
```


Now our crawler knows what links to follow. We know that amazon, like most of modern sites, uses javascript to display content. In some cases it makes scraping much more complicated but it's a good thing that amazon works perfectly without any javascript so we don't have to use any kind of head-less browser or such.

Scrapy FormRequest

As I said, we want to scrape the most reviewed books in each category, let's say the first 12 we can find on the first page. But first we should sort the books by most reviews.

```
<form id="searchSortForm" class="s-inline-form" method="get" action="/gp/search/ref=sr_st">
  <input name="rh" value="n:283155,n:!1000,n:1" type="hidden">
  <input name="qid" value="1458242511" type="hidden">
  <span class="a-size-base">Sort by </span>
  <select id="sort" class="a-spacing-top-mini" style="vertical-align: baseline;" name="sort" onchange="">
    <option value="featured-rank" selected="selected">Featured</option>
    <option value="price-asc-rank">Price: Low to High</option>
    <option value="price-desc-rank">Price: High to Low</option>
    <option value="review-rank">Avg. Customer Review</option>
    <option value="date-desc-rank">Publication Date</option>
    <option value="review-count-rank">Most reviews</option>
  </select>
  <noscript><input type="image" src="http://g-ecx.images-amazo...</noscript>
</form>
```

If you have a look at the source you'll see that we need to parse a form in order to sort. The "Go" button which will refresh the page according to the form is visible only if visiting the page without javascript enabled like our crawler does. In Scrapy we can use a [FormRequest](#) object to pass a form:

```
def sort_books(self, response):
    sorted_form = FormRequest.from_response(
        response,
        formxpath="//form[@id='searchSortForm']", #xpath of form
        formdata={"sort": "review-count-rank"}, #id of select node("sort") and
        value #of "Most Reviews"("review-count-rank")
        clickdata={"value": "Go"}, #the button we "click"
        callback=self.parse_category #the method we execute on the sorted
        page
    )
    yield sorted_form
```

Data Extraction

The next thing we have to do is to parse each link that redirect the crawler to a book's page where you invoke the *parse_book_page* method which will take care of scraping the data we're looking for.

```
def parse_category(self, response):  
    links = response.xpath("//a[@class='a-link-normal s-access-detail-  
page a-text-normal']/@href")  
    for link in links:  
        url = response.urljoin(link.extract())  
        yield Request(url, callback=self.parse_book_page)
```

Finally, we extract the desired details we need in *parse_book_page*.

Exporting Json and CSV in Scrapy

The real beauty in web scraping is actually to be able to use the scraped data. In most cases, the easiest and smartest way to store scraped data is a simple Json or CSV file. They are readable by humans and other softwares as well so it should be applicable almost everytime though when you work with huge amount of data it might be better to choose a database structure which is more scalable.

There are some ways to produce Json or CSV files including your data in Scrapy.

The first way is to use Feed Exports. You can run your scraper and store your data from the command line by setting the filename and desired format.

You may want to customize your output and produce structured Json or CSV while your scraper runs. You can use Item Pipeline to set your output properties in a pipeline and not from command line.

Exporting with Feed Export

As you learnt in the first Scrapy tutorial you can run your scraper from command line with the `scrapy crawl myspider` command. If you want to create output files you have to set the filename and extension you want to use.

```
scrapy crawl myspider -o data.json
```

```
scrapy crawl myspider -o data.csv scrapy crawl myspider -o data.xml
```

Scrapy has its built-in tool to generate json, csv, xml and [other serialization formats](#).

If you want to specify the path of the produced file or set [other properties](#) from command line you can do it as well.

```
scrapy crawl reddit -s FEED_URI='/home/user/folder/mydata.csv' -s  
FEED_FORMAT=csv
```

```
scrapy crawl reddit -s FEED_URI='mydata.json' -s FEED_FORMAT=json
```

Exporting with Item Pipeline

Scrapy [Item Pipeline](#) is a universal tool to process your data. Typical usages are cleaning html, validating scraped data, dropping duplicates and storing scraped data in database. You can use pipelines if you want a convenient and customizable way to store your data.

Exporting data to json is pretty simple. Import [python json package](#) then transform your item to json format and print it to a json file.

```
class JsonPipeline(object):

    def __init__(self):
        self.file = open('pipelinejson.json', 'wb')

    def process_item(self, item, spider):
        line = json.dumps(dict(item)) + "\n"
        self.file.write(line)
        return item
```

It works the same way with CSV but you have to import [python csv module](#).

```
class CsvPipeline(object):

    def __init__(self):
        self.writer = csv.writer(open('pipelinecsv.csv', 'a'),
lineterminator='\n')

    def process_item(self, item, spider):
        csv = [item[key] for key in item.keys()]
        self.writer.writerow(csv)
        return item
```

Configure settings.py

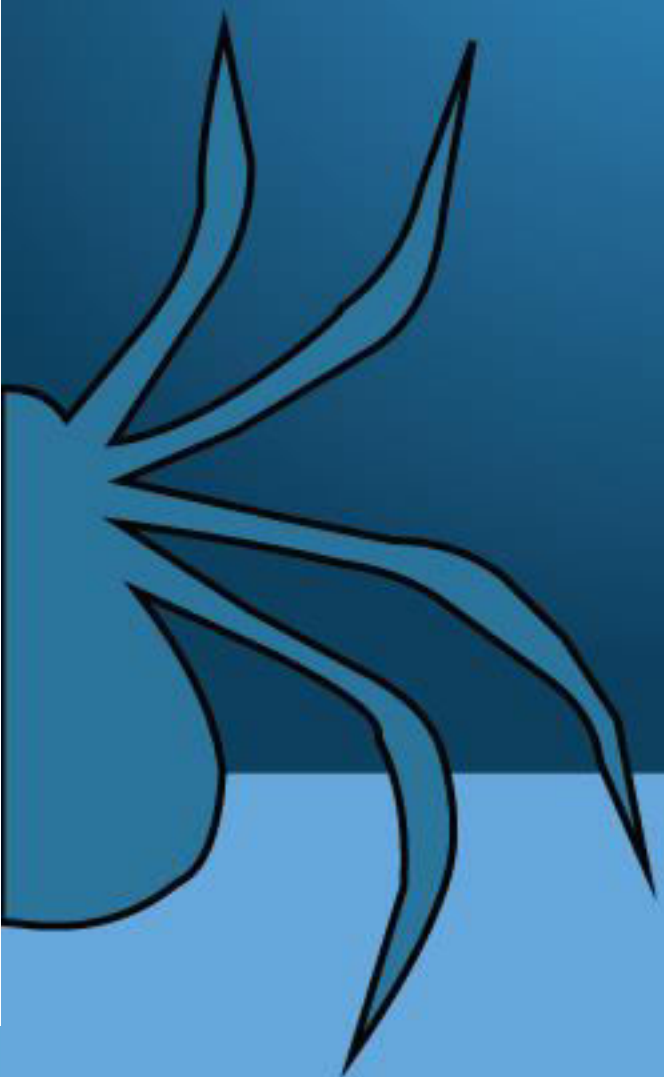
It's very important to tell scrapy you dare to use Item Pipelines otherwise your pipelines won't be invoked.

You have to add these lines to your settings.py in your Scrapy project:

```
ITEM_PIPELINES = {
    'RedditScraper.pipelines.JsonPipeline': 300,
    'RedditScraper.pipelines.CsvPipeline': 500,
}
```

If you are wondering what those numbers mean, those are meant to indicate the priority of a pipeline. The higher the number the sonner it will be run.

Ethical Web Scraping



Ethical and Legal Web Scraping

As web scraping is getting more and more popular it's necessary to speak about it in perspective of laws and ethics. So I decided to do comprehensive research on this and with my web scraping experience create a guide which you can follow to build a scraper that is lawful and ethical. Making your life much easier.

Before 2000, web scraping was a gray area in the legal system of US. There was no significant precedent around web scraping. The first time a company was sued for web scraping related activities happened on December 10, 1999, [Ebay v. Bidder's Edge](#).

Bidder's Edge was an aggregator of auction listings. Users could search through tons of auction websites, including Ebay, without visiting each individual site. Bidder's Edge accessed eBay approximately 100,000 times a day, it was 1.53% of the whole traffic. When Ebay got the case to the court they claimed that Bidder's Edge without authorization interfered with eBay's possessory interest in the computer system and resulted in damage to Ebay. In March 2001, they settled their legal disputes: Bidder's Edge paid Ebay an undisclosed amount and agreed not to access Ebay's data.

Copyright Infringement

You are probably already familiar with this. How does it relate to web scraping? Actually it doesn't. The point here is what you do with the data after you scraped it. If it is copyright-protected - in most cases it is - then you can't do whatever you want with the data. For example, you cannot use it for commercial purposes or you cannot upload it to your own site. So next time you scrape some website make sure you are allowed to use the scraped information the way you want to.

Violation of the Computer Fraud and Abuse Act (CFAA)

This law wasn't invented to prevent web scraping as well. Actually it is mainly against hackers. In short, this one is about fetching data by getting unauthorized access to a page. Considering that using web scraping techniques you can only reach data that are publicly available we would think we have nothing to do with this law.

Though, it is also true that some scrapers, taking advantage of people or making fun of them, can violate this law. This was exactly [how Jerk.com worked](#) back then in 2009. It stole personal photos from facebook then asked for money to remove it. It's really unethical and unlawful.

Trespass to Chattel

This one is really easy to forget about. At least from a scraper perspective. You violate this law if you hurt directly the website server in any way. Actually, as a web scraper, it's an easy thing to do. A typical mistake that almost every novice scraper do: they make requests relatively too often. At first, we don't care about the number and frequency of http requests. We care about only getting the data we need as soon as possible. It's not a good practice!

Having 1000 request of various pages per second can really decrease the performance of the site. You violate trespass to chattel if you make the server slow or even stop because of your scraper. Or if your scraper does something that distracts the natural life of the website you do something wrong! And even worse that can happen is the website owner could think that you're doing harm intentionally to the website by requesting pages with high frequency. It may seem like you try to attack the website. Be careful!

Disclaimer

I'm supposed to tell you some important statements. I'm not a lawyer. I don't even know any lawyers. This whole post is based on my web scraping experience and a comprehensive research on the topic. I hope you get some value out of it.

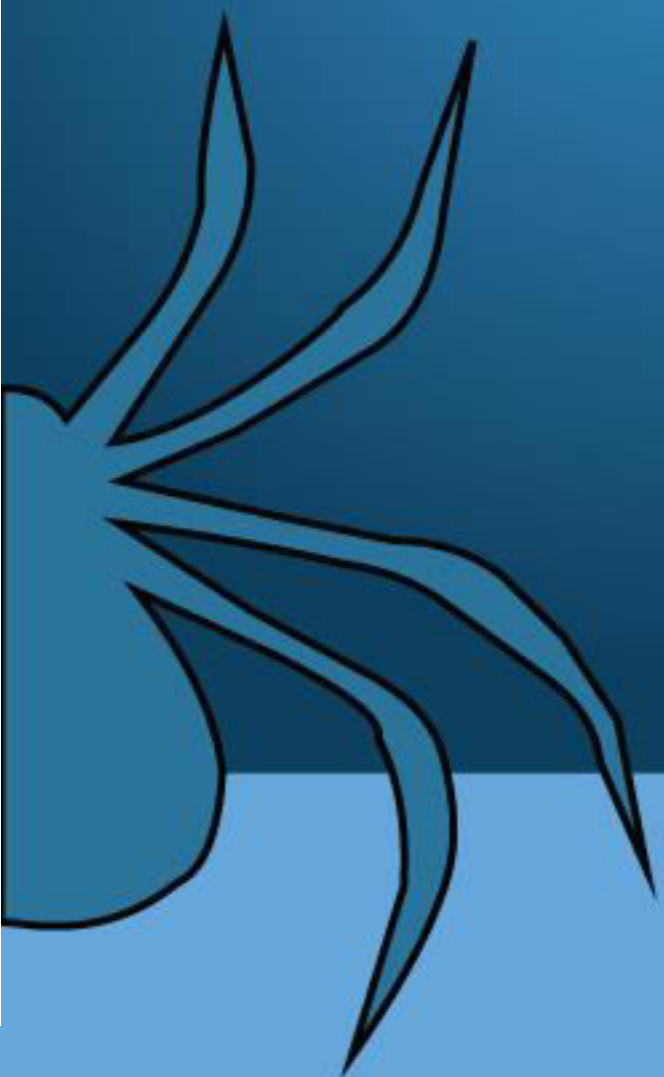
How to Build an Ethical Web Scraper

Lately, the [ScrapingHub blog](https://blog.scrapinghub.com/2016/08/25/how-to-crawl-the-web-politely-with-scrapy/) released an awesome article about how you can build an ethical and lawful web scraper. I really suggest that you should check it out. Here's the link:

<https://blog.scrapinghub.com/2016/08/25/how-to-crawl-the-web-politely-with-scrapy/>



Where to Go Next



Contact

I write blog posts every week about web scraping, subscribe to my email list to keep up: <http://www.scrapingauthority.com/signup>

If you have anything to say or ask me, shoot me an email here:

attila@scrapingauthority.com

My Twitter: <http://twitter.com/scrapingA>

My Facebook: <http://facebook.com/scrapingauthority/>

Share

Did you like this book?

Share it with your friends!

Thank you for reading my book I hope you liked it.

You can share this book with this link:

At the end, I'd like to tell you that I'm in the making of a **package for fellows who want to delve deep into Web Scraping with Scrapy Framework**. The package will contain **video tutorials, screencasts** and other exclusive contents to teach you web scraping with Scrapy.

Make sure to **signup** to my email list to get noticed first when I publish it: <http://www.scrapingauthority.com/signup>