How to create a basic web crawler with Scrapy

Writing these web crawler programs is easier than you think. Python has an excellent library for writing scripts that extract information from web pages. Let's see how to create a web crawler with Scrapy through the following article!

Programs that read information from a web page or web crawler, have all sorts of useful applications.

Writing these web crawler programs is easier than you think. Python has an excellent library for writing scripts that extract information from web pages. Let's see how to create a web crawler with Scrapy through the following article!

Use Scrapy to create web crawlers

1. Install Scrapy
2. How to create a web crawler
    1. Turn off logging
    2. Use Chrome Inspector
    3. Extract title
    4. Find description
3. Collect JSON data
4. Exploit many factors

Install Scrapy

Scrapy is a Python library created to scan and build web crawlers. It is fast, simple and can navigate through many websites without a lot of effort.

Scrapy is available through the Pip Installs Python library (PIP). For how to install PIP, please refer to the article: Installing Python Package with PIP on Windows, Mac and Linux.

Using the Python virtual environment is preferred because it will allow you to install Scrapy in a virtual directory and keep the file system intact. The Scrapy documentation recommends doing this for best results.

Create directories and initialize a virtual environment.

```
mkdir crawler cd crawler virtualenv venv . venv/bin/activate
```

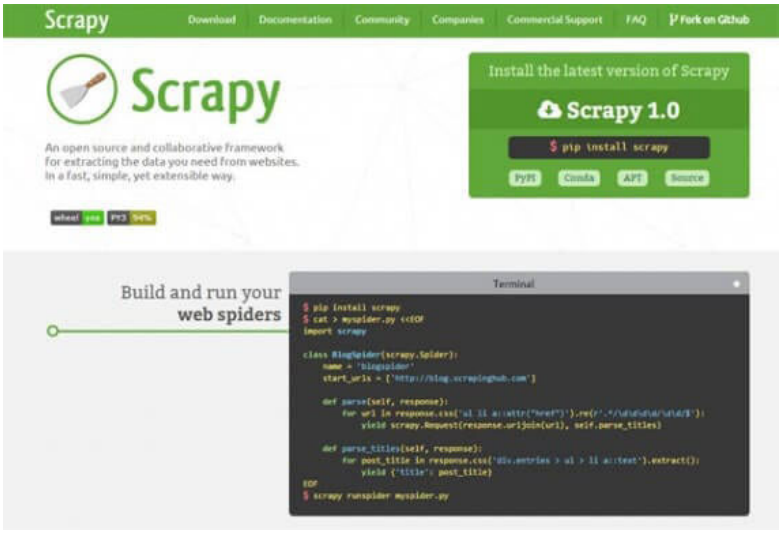
Now you can install Scrapy into that directory using the PIP command.

```
pip install scrapy
```

Quick check to make sure Scrapy is properly installed:

```
scrapy # prints Scrapy 1.4.0 - no active project Usage: scrapy [options] [args] A
```

```
scrapy # prints Scrapy 1.4.0 - no active project Usage: scrapy [options] [args] A
```



Scrapy

How to create a web crawler

Now that the environment is ready, you can start creating web crawlers. Take a look at some information from the Wikipedia page about batteries:

```
https://en.wikipedia.org/wiki/Battery_(electricity)
```

The first step to writing a crawler is to define the Python class extending from **Scrapy.Spider.** This gives you access to all functions in Scrapy. Call this class **spider1.**

The spider class needs some information:

1. Name (**name**) to identify the spider
2. The **start_urls** variable contains a list of URLs to crawl (Wikipedia URLs will be examples in this tutorial)
3. The **parse () method** is used to process web pages and extract information.

```
import scrapy class spider1(scrapy.Spider): name = 'Wikipedia' start_urls = ['ht
```

A quick test to make sure everything is running properly.

```
scrapy runspider spider1.py # prints 2017-11-23 09:09:21 [scrapy.utils.log] INFO
```

Turn off logging

Running Scrapy with this class will export log information which will not help at the moment. Make things simple by removing this redundant log information. Add the following code to the beginning of the file.

```
import logging logging.getLogger('scrapy').setLevel(logging.WARNING)
```

Now when you run the script again, the log information will not be printed.

Use Chrome Inspector

Everything on a web page is stored in HTML elements. Elements are organized in the Document Object Model (DOM). Understanding the DOM is important to make the most of the web crawler. Web crawlers search through all the HTML elements on a page to find information, so it's important to understand how they are organized.

Google Chrome has tools to help you find HTML elements faster. You can position the HTML in any element you see on the web with the inspector.

1. Navigate to a page in Chrome
2. Place your mouse over the element you want to see
3. Right click and select **Inspect** from the menu

These steps will open the developer console with the **Elements** tab selected. At the bottom of the console, you will see a tree diagram containing the elements. This tree diagram is how you will get information for the script.

Extract title

Use the script to do some work. Gather simple information to get the site's title text.

Start the script by adding some code to the parse () method to extract the title.

```
. def parse(self, response): print response.css('h1#firstHeading::text').extract
```

The **response** argument supports a method called **CSS () that** selects elements from the page using the location you provide.

In this example, the element is **h1.firstHeading.** Adding **:: text** to the script is what is needed to give you the content of the element. Finally, the **extract () method** returns the selected element.

Run this script in Scrapy to export the title in text form.

```
[u'Battery (electricity)']
```

Find description

We have now extracted the title text. Do more with scripts. The crawler will find the first paragraph after the title and extract this information.

Here is an element tree diagram in the Chrome Developer Console:

```
div#mw-content-text>div>p
```

The right arrow (>) indicates the parent-child relationship between the elements. This position will return all matching **p** elements, including the entire description. To get the first **p** element, you can write this code:

```
response.css('div#mw-content-text>div>p')[0]
```

Like the title, you add :: text to get the text content of the element.

```
response.css('div#mw-content-text>div>p')[0].css('::text')
```

The final expression uses **extract ()** to return the list. You can use the Python **join ()** function to concatenate lists after the crawl is complete.

```
def parse(self, response): print ''.join(response.css('div#mw-content-text>div>p
```

The result is the first paragraph of the text!

```
An electric battery is a device consisting of one or more electrochemical cells w
```

Collect JSON data

Scrapy can extract information in text form, very useful. Scrapy also allows you to view JavaScript Object Notation (JSON) data. JSON is a neat way to organize information and is widely used in web development. JSON also works quite nicely with Python.

When you need to collect data in JSON format, you can use the **yield** statement built into Scrapy.

Here, a new version of the script uses the **yield** statement **.** Instead of taking the first **p** element in text format, this statement will retrieve all the **p** elements and arrange it in JSON format.

```
. def parse(self, response): for e in response.css('div#mw-content-text>div>p'):
```

Now, you can run the **spider** by specifying the output **JSON** file:

```
scrapy runspider spider3.py -o joe.json
```

The script will now output all **p** elements.

```
[ {"para": "An electric battery is a device consisting of one or more electroche
```

Exploit many factors

So far, the web crawler has tapped the title and a type of element from the page. Scrapy can also extract information from different types of elements in a script.

Feel free to extract the top IMDb Box Office hits for the weekend. This information is taken from *http://www.imdb.com/chart/boxoffice* , in a table with rows for each metric.

The **parse () method** can extract multiple fields from a row. Using Chrome Developer Tools, you can find nested elements in the table.

```
. def parse(self, response): for e in response.css('div#boxoffice>table>tbody>tr
```

The selector **image** identifies that **img** is a descendant of **td.posterColumn.** To extract the correct attribute, use the expression **:: attr (src).**

Running spider returns JSON:

```
[ {"gross": "$93.8M", "weeks": "1", "weekend": "$93.8M", "image": "https://images
```