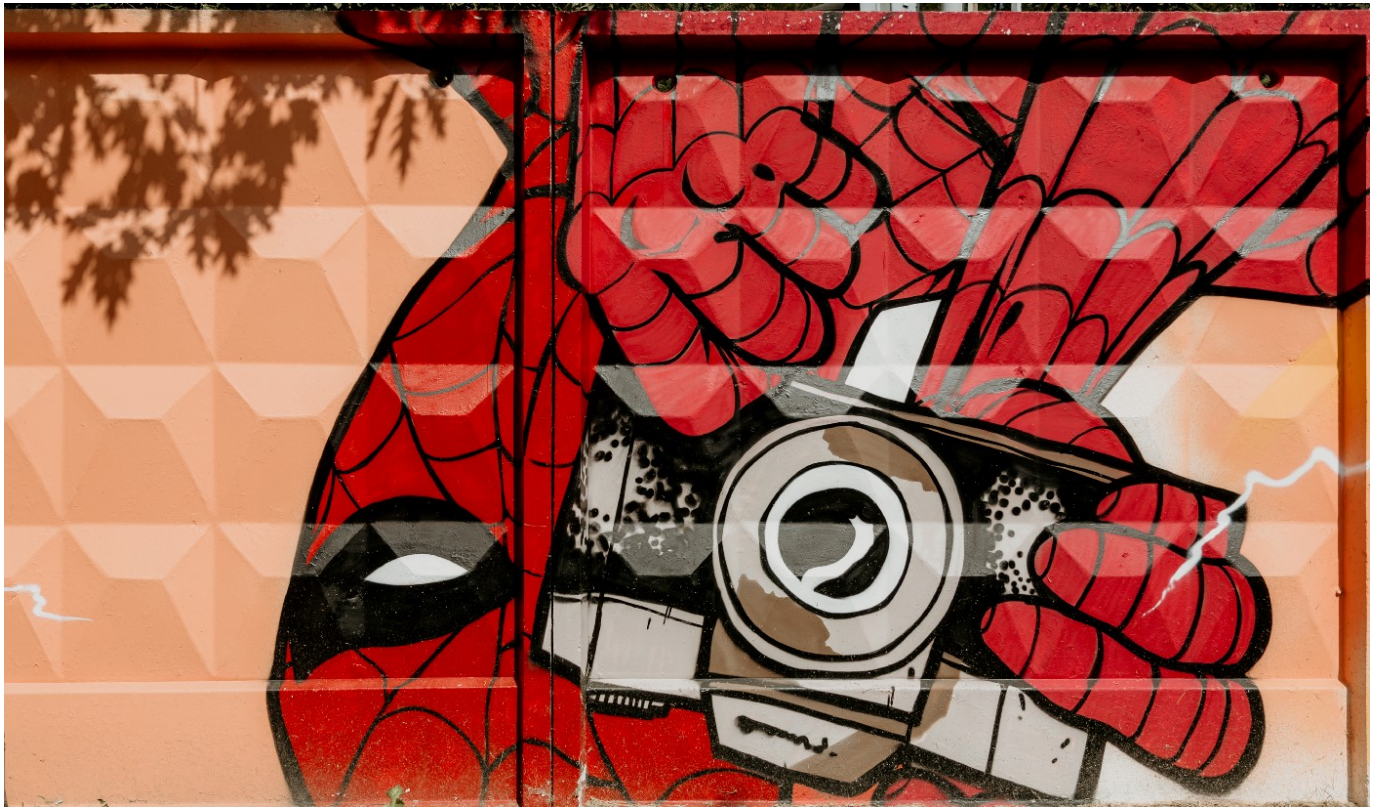


Photo by Marjan Blan | @marjanblan on Unsplash

Web Scraping 2.0

Over The Top Web Scraping Using Scrapy

Abhay Parashar Follow

Feb 10 · 12 min read ★

Scrapy is a full-stack python framework for web scraping. It is a tool for large-scale web scraping. It has a built-in mechanism called selectors for extracting data from the web. It is an open-source and free-to-use framework written in python. It can automatically control the crawling speed using the autothrottle mechanism. Scrapy scrapes data from the website with the help of spiders. It can crawl through the entire website in a systematic way in just a few minutes. let's look at some of the features of it.

Features:-

- Low CPU and Memory Usage.
- Easy To Follow Documentation.
- Can Scrap multiple websites at the same time.

- Asynchronous
- Uses Spiders To Crawl Websites.

Installation: `pip install scrapy`

Key Components of Scrapy:-

1. Scrapy Shell

Scrapy provides an interactive shell that can be used to debug and test your scraping code very quickly without having to run the spider. spiders are classes that define how a site will be scrapped. scrapy shell is a type of python shell that means you can run and test your python scripts here in the shell too.

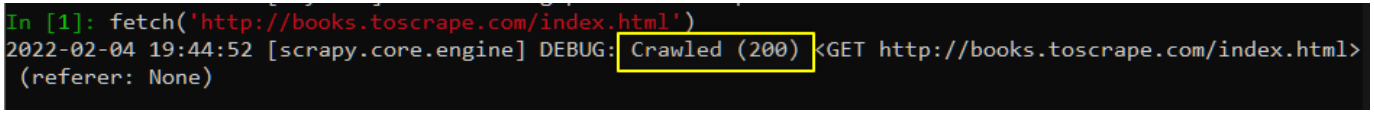
It is mostly used for testing the Xpath and CSS expressions to check whether they are working or not.

After Installing Scrapy you can launch this shell by the following the command —

```
scrapy shell

OR

scrapy shell "URL"
```

Once the shell opens up you can use it to scrape any data from the web. To send a crawling request to a web server we use `fetch(URL)`

```
fetch(URL)

-----
fetch('http://books.toscrape.com/index.html')
```



After Running the above fetch command you will see a debug message Crawled(200) means your website is working and the connection request is successful.

In scrapy the source code of the website is stored inside a variable `response` , you can use it to extract data by passing a selector expression inside it.

Let's scrape the title of the page using the shell —

```
response.css("title")


----------------
[<Selector xpath='descendant-or-self::title' data='<title>\n    All
products | Books to S...'>]
```

When You Run The Command a selector list gets returned as output that contains the particular CSS element you are requesting for.

Now To Scrape The Tag From This List We use `extract()`

```
response.css("title").extract()


----------------------------
['<title>\n    All products | Books to Scrape - Sandbox\n</title>'
```

The returned output contains the tag inside a list. To fetch the text from it we use `::text` and specify it at the end of the selector expression.

```
response.css("title::text").extract()

-------------------------
'\n    All products | Books to Scrape - Sandbox\n'
```

Now to clear the test even more we can use string inbuilt functions like strip() for removing whitespace and replace for replacing a keyword with something else.

```
response.css('title::text').extract()[0].strip().replace('\n',' ')

-----------------
'All products | Books to Scrape - Sandbox'
```

Next, we will try to scrape all the titles of books using the shell itself.

HTML Snippet of First Book Title

The Title of the first book is inside an anchor `<a>` tag that is the child of a heading 3 `<h3>` tag. To scrape it we will first target the <h3> tag and then the <a> tag to fetch text from it. The HTML structure is the same for all the titles on the page, so if we scrape one all the title gets scraped automatically. The scraped data is stored inside a list.

```
response.css("h3 a::text").extract()
```

The above code will scrape all the book titles from the web page. Try to remove white space and line breaks using the string functions on your own.

You Can Use this shell to debug and test some lines of code inside your big scraping projects. Now, Let's move on and create a project.

. . .

2. Project Structure

Scrapy is a complete web scraping framework and follows a systematic approach for scraping data. There is a proper project structure that scrapy follows.

You can start a new project in scrapy with the following command.

```
scrapy startproject PROJECT_NAME

------
scrapy startproject bookscraper
```

After running the command your project structure will get created which looks something like the below —

Project Structure

There is only one folder you need to care about for now is `spiders` folder, where you will put all your spiders, means scraping codes later.

. . .

3. First Spider

Open the spiders folders and create a python file named `book_scraper.py` and open it. The Spiders folder is where you will put all your spiders. you can create multiple spiders for scraping different content and connect them together.

Let's Borrow an example from the official docs to understand everything better —

```python
import scrapy
class QuotesSpider(scrapy.Spider):
    name = "quotes"
    def start_requests(self):
        urls = [
            'http://quotes.toscrape.com'
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)
    def parse(self, response):
        title = response.css('title::text').extract()
        print(title)
```

Let's Understand Above Code Line By Line

Line 1: Importing The Library.
Line 2: We will create a class that is used by scrapy to scrape the content from the web. You can name it anything but the important thing is that it will inherit the class Spider.
Line 3: `name` is the name of our spider. You can again give it any name but try to not add any whitespace between text because this name is going to be used when you run your spider.
Line 4: It is the function we use to define all the URLs inside our spider.
Line 5: `URLs` is a list of URLs to be scraped.
Line 8,9: A for loop that runs on the URLs list and extracts each URL one by

one and passes it to scrapy for scraping data.
Line 10: `parse` is a method of class that takes two inputs one is `self` and other is `response` that contains the source code of the website you want to scrape. you can rename it too.
Line 11: In this line, we are trying to scrape the title of the web page using a CSS selector.
Line12: We are simply printing the title.

To Run the above code run the following command —

```
scrapy crawl quotes
```

In command, `scrapy` is the library essential, `crawl` is an initiator for scraping and `quotes` is the name of your spider that you have initialized while writing your web scraping code. It will return all the titles inside a list.

Let's understand more deeply about these selectors in the next section.

. . .

4. Element Selectors

There are two ways to select an element on the web while using scrapy — CSS and Xpath.

1. CSS Selectors

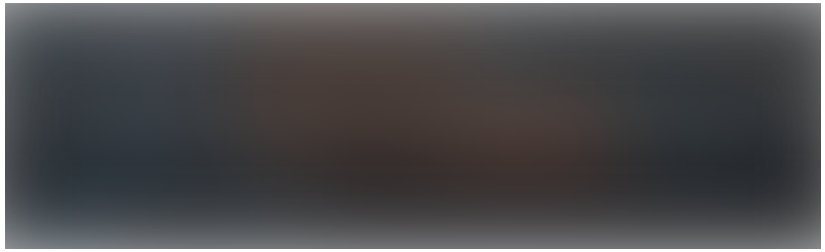
The condition inside `css()` is called a CSS selector. Let's take a look at some examples.

Open The Scrapy Shell and Write The Following Code to establish a connection between shell and server. we are going to Scrape book names and prices of all the books on the website books to scrape using CSS selectors.

```
scrapy shell "http://books.toscrape.com/index.html"
```

Price is the first thing We Wanted to scrape of all the books. Open the website, Hover over a price, and then right-click> inspect to open the

source code for that element.



HTML Snippet For First Book Price

Our First Book Price is inside a paragraph tag `<p>` that is a child of an div tag. The `<p>` has a class of `price-color` . The Unique identifier for the price will be a div that has a p tag as a child with a class name as `price-color` , let's use it to scrape all the prices.

```
response.css("div p.price_color::text").extract()
```

we are using a `.` before the `price_color` to represent it as a class. if it was an id then we should have used `#price_color`

Next, we will scrape all the titles of the book with the help of the following code

```
response.css("h3 a::text").extract()
```

2. Xpath

Xpath is also a way of locating elements on XML documents. HTML is an implementation of XML so we can use it to locate elements in HTML as well.

The Basic Syntax For an Xpath is —

```
Xpath = //tagname[@Attribute='Value']

//        ➞ Select Current Node
tagname   ➞ Tagname like input, div,td,tr
@         ➞ Selects attribute
Attribute➞ Attribute name (class,id,name,etc)
value     ➞ value of the attribute
```

Let's use Xpath to extract the titles and prices of the books.

```
## Scraping Titles
titles = response.xpath('//h3/a/text()').extract()

## Scraping Prices
Prices =
response.xpath('//div/p[@class="price_color"]/text()').extract()
```

Xpath is a little different from CSS, here we use `/text()` to extract the text from the scraped text.

Let's understand the Xpath for prices, first, we have `//div` means the top level is a div tag, then we have `//div/p` means inside a div tag we have a paragraph tag. at last, we have `//div/p[@class="price_color"]` that targets the scrapy to locate for a paragraph element that has a class of price_color and child of a div tag.

. . .

You can use any method out of two for scraping data from the web. After running the code two lists `Titles, Prices` returned as output that contains

titles and prices for the first page.

But what if we want titles and prices from every page of the website. here comes the concept of multipage scraping.

4. Multipage Scraping

Multipage scraping is the process of scraping multiple pages of a website at once by manipulating the URL.

The website we are working on is http://books.toscrape.com/index.html ,

if you scroll to the bottom you will see the next button.

By clicking on it you get redirected to page 2 and the URL of the website changes to *http://books.toscrape.com/catalogue/page-2.html*

now again scroll till bottom click on next and check the URL it gets updated to *http://books.toscrape.com/catalogue/page-3.html*

now do it again, and check the URL this time it gets updated to *http://books.toscrape.com/catalogue/page-4.html*

you can notice there is only one thing that is changing after the second page is the page number in the URL. we can use it to create a universal URL by replacing them `page-2` with an f-string variable.

```
for page_number in range(50):
  url=f'http://books.toscrape.com/catalogue/page-{page_number}.html'
  print(url)
```

This method is a universal method that you can use even if the next method doesn't work. Now, let's look at a simpler method.

Just scroll down till the end and inspect the `next` button on your home page.

`<a>` tag contains the link to the second web page. Now, go to the second page and inspect the next button you will see that this time it has the link for page 3. All we need is to scrape the link from this tag and everything else scrapy will do for us.

To Scrape the link firstly we target the button and then extract the link from it using the XPath `//li[@class='next']/a/@href` , `@` is used to extract the value of an attribute from the tag.

Let's use this to scrape all the web pages.

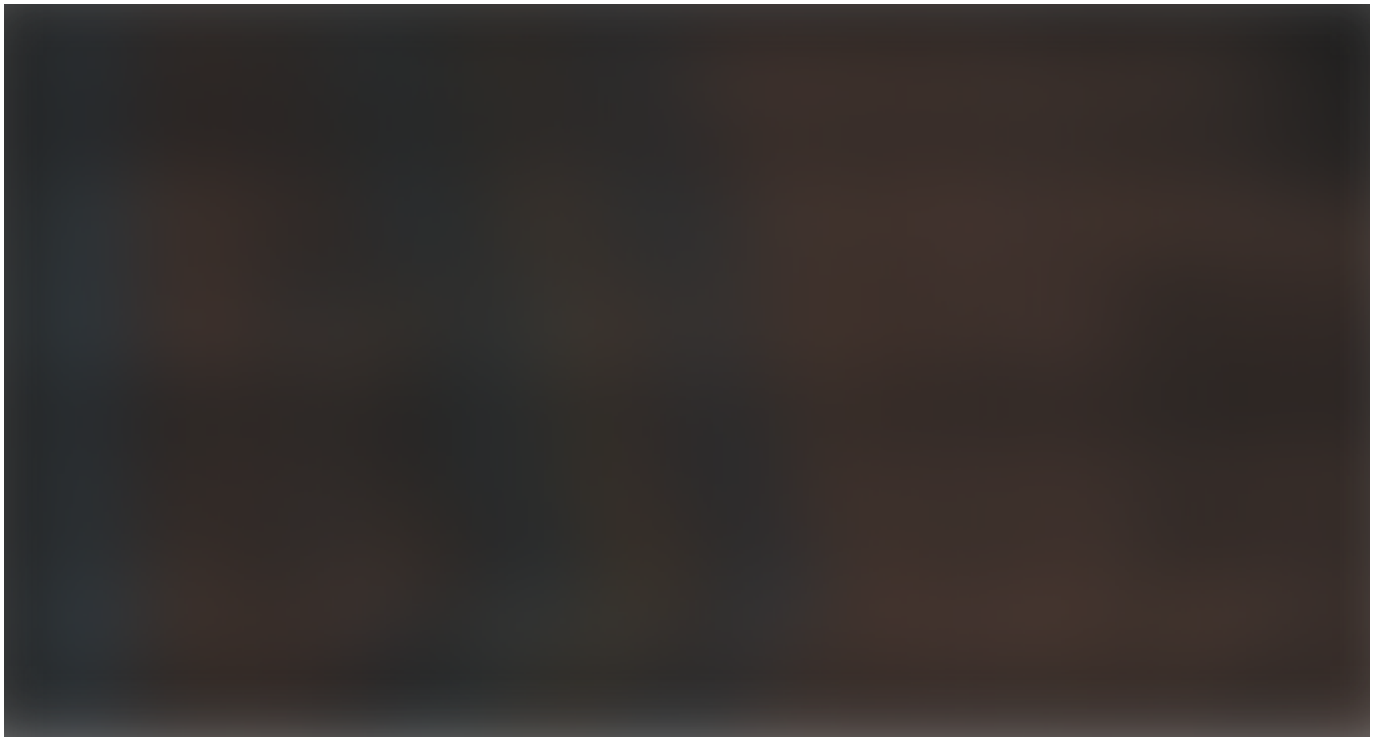
Run the Above Code using the following command

```
scrapy crawl books
```

Scraped Data Output Gif By Author

You can save the scraped data in a CSV, JSON, or XML file by running the following command at the start —

```
 scrapy crawl books -o data.csv
or
 scrapy crawl books -o data.json
or
 scrapy crawl books -o data.xml
```



data.json file screenshot by the author

. . .

5. Useful Concepts

1. Item Container

The main goal of web scraping is to scrape unstructured data from the web and store it in a structured manner. These item containers are used to store the data. Scrapy by default returns the data as items.

items provide a dictionary-like API that can be used to convert unstructured data into structured data. `items.py` file is used to define containers inside a scrapy project.

The `items.py` file is created automatically when you create a project using scrapy. You need to import it on your spider to make use of item containers as we did in line 2 of the above book scraping code.

```
from ..items item BookscraperItem
```

BookscraperItem is the class name that is automatic gets created when you start a project. The name of the class depends on your project name. It is used to define your item container.

```
items.py


class BookscraperItem(scrapy.Item):
# define the fields for your item here like:
# name = scrapy.Field()
    link = scrapy.Field()
    price = scrapy.Field()
    title = scrapy.Field()
```

Once containers get defined you can use them to store your data. Remember one thing the name of the container should be the same as the name of your data variables.

2. Parse

`parse` is a method that is in charge of processing the response and returning scraped data or more URLs to follow. It takes the `response` variable as input that contains the source code. This method is also used to pass the data to item containers for temporary storage.

By default, the code will pass the callback to the parse method until you specify something else. like we did in line 13 of the book scraping code.

```
def parse(self,response):
      ...
```

. . .

Project 1: Amazon Books Scraping

Task: Our task is to scrape all the books titles, author names, and prices for python programming from amazon.

Step 0: Create a Project

Run `scrapy startproject amazon` to start a new project and generate the folder structure.

Next, Target to spiders folder and create a new python file `scraper.py` for writing web scraping code.

Step 1: Importing Libraries

```
import scrapy
from ..items import AmazonItem  ## class inside items.py
```

Step 2: Creating Class & Naming Spider

```
class Amazonbookscraper(scrapy.Spider):
   name = "amazonbooks"
```

Step 3: Preparing Starting URLs

```
start_urls = [
"https://www.amazon.com/s?
k=python+programming&page=1&crid=YZEI0VGEO20K&qid=1644079860&sprefix=
python+programming&ref=sr_pg_2"
]
```

`start_urls` is a list that contains a list of URLs to be scraped. It is a simpler way of putting your URLs for scraping.

Step 4: Scraping Data

1. Scraping Titles

There are Three Classes You Can Use Any

```
titles = response.css('.a-size-base-plus::text').extract()
```

2. Scraping Prices

a-price-whole class contains the information about the price of the books

```
prices = response.css('.a-price-whole::text').extract()
```

Now, To scrape Authors and Ratings Let's use the **Selector Gadget chrome extension**. Check out this **youtube video** to know how to use it to find CSS or XPath of a particular element on the web page.

3. Scraping Authors

```
authors = response.css('.a-color-secondary .a-size-base+ .a-size-
base::text').extract()
```

4. Scraping Ratings

```
ratings = response.css('.s-link-style .s-underline-
text::text').extract()
```

Step 5:Storing Data Inside Item Containers

```
for i in list(zip(titles,prices,authors,ratings)):
  title,price,author,rating = i
  items['title']=title
  items['price']=price
```

```
    items['author']=author
    items['rating']=rating


  yield items
```

Step 6: Scraping Multiple Pages

```
next_page_url = "https://www.amazon.com"+response.xpath('//*
[contains(concat( " ", @class, " " ), concat( " ", "s-pagination-
next", " " ))]/@href')[0].extract()


if next_page_url is not None:
    yield response.follow(next_page_url, callback=self.parse)
```

Full Code

Run The Code Using the command `scrapy crawl amazonbooks`

You might be stuck into a connection error from Amazon. It is because Amazon is blocking your connection-making requests. To prevent this we

can make use of User Agents. There is a library created by the lovers of scrapy that contains over 2000 User Agents and changes user agents frequently so that you don't get blocked. you can install this library using the following command.

```
pip install scrapy-user-agents
```

Once it gets installed, go to your `settings.py` file in your project folder. Paste the following code below the download malware comment.

```
DOWNLOADER_MIDDLEWARES = {
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware':
None,
    'scrapy_user_agents.middlewares.RandomUserAgentMiddleware': 400,
}
```

Now, Again try to run the code. This time it will run successfully and scrape the data from amazon.

Amazon Book Scraper Output

Books_details.json File

. . .

Recommended Readings

Master Web Scraping Completely From Zero To Hero

Using Beautiful Soup and Requests Library with One Project

medium.com

Master The Art of Writing Xpath For Web Scraping

A Gentle Introduction To The RegEX of Web Scraping

levelup.gitconnected.com

Web Scraping Using Selenium Python

Detailed Tutorial With One Project

medium.com

. . .

Thanks For Reading Till Here, If You Like My Content and Want To Support Me The Best Way is —

1. Follow Me On *Medium*.
2. Connect With Me On *LinkedIn*.
3. Become a Medium Member Using *My Referral Link*. a small part of your membership fee will go to me.
4. Subscribe To *My Email List* To Never Miss An Article From Me.

A monthly summary of the best stories shared in Level Up Coding Take a look.

Get this newsletter

Web Scraping | Python | Data Mining | Education | Data Science

82

WRITTEN BY

Abhay Parashar

Top Writer | Engineer | Learning and Sharing Knowledge Everyday| Editor of The Pythoneers | Become a medium member bit.ly/3l3PMj4 ☺

Follow

Level Up Coding

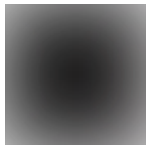
Coding tutorials and news. The developer homepage gitconnected.com && skilled.dev

Follow

More From Medium

How to create a modular system for powerups

Eero Saarinen



Builder Pattern in Scala with Phantom Types

Maximiliano Felice

My Experience of preparing for AZ 900 (Azure Fundamentals) Cloud Certification

Saurav Das

Defining Custom Units in Julia and Python

Erik Engheim

Serverless framework — Building Web App using AWS Lambda, Amazon API Gateway, S3, DynamoDB and...

Andrei Maksimov

BRT CSS

RGPV Objective question For online Mode exam

Reward for 100 LIGHT 😈

Airdrop Finder

My First Pull Request to OpenFaaS — a major open-source project!

Burton Rheutan in HackerNoon.com