**OFFICIAL NOTES:**

**What is Scrappy:**

\*Scrapy is an open-source library that we can use in python, what scrapy does is that it allows us to scrap data from a website, and then to format the data that we scrapped off the website into a neat format, that we can easily perform data analytics on.

\*so, remember that in a nutshell data analytics is all about our ability to clean data, and once we have cleaned the data we are then able to process the given data and change it into useful information that is able to aid us in our decision making processes. But how do we collect the data, that’s where data scraping comes into the picture, here we need to know how to scrap data from a given website, and once we have scraped the data we need to be able to structure the data into rows and columns so that we can be able to clean the data.

------------------------------------------------------------------------------------------------

Part 4 : Building your first Scrapy Spider



1. We need to go and create a Scrapy spider

\*A scrapy spider is a program that is designed to systematically crawl through web pages and extract data from the webpages. It also has the ability to crawl through other webpages that are liked to the given webpage and also extract data from those web pages.

**How does a scrapy spider work:**

\*immediately after we create the Scrapy spider called “Bookspider” we should see the following file being created, and it will have the following code by default.

**A computer screen with text and images

Description automatically generated**

Step 1) Initialize the spider (line 4) :



\*we are creating a class here, and we know that a class a blueprint for creating Objects. We are creating a class called” BookspiderSpider” and this class is able to inherit from scrapy.Spider.

\*But we know that in Python a class is created using Upper case letters, the first letter of a class will always be in upper case letters, hence when the class name is created Python, creates it using an upper case letter.

Where:

Bookspider is the name of our Spider, we chose this name when we created the spider using the following command

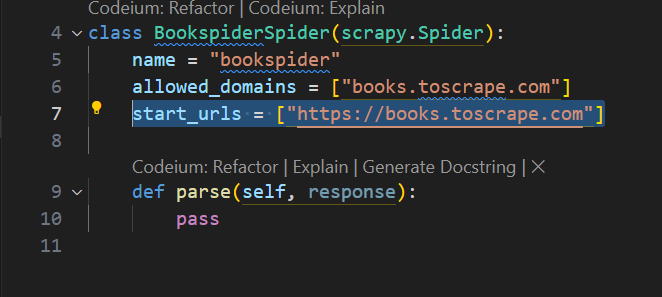
|  |
| --- |
| $ scrapy genspider bookspider books.toscrape.com  \* bookspider is the name of the spider  \* books.toscrape.com is the name of the website that we want to scrap |

Spider is what we call the base class, the base class contains modules, and in this case the module that we want to inherit from this base class is scrapy.

Scrapy is the module that we want to inherit, it’s the module that we import, and its also the module that we want to inherit, and we want to inherit it from the base class with is Spider

**Step2** : define the start URL’s

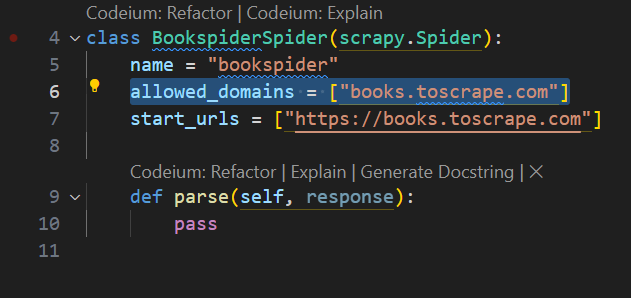
**Line 7** :



\*we need to go and define an attribute called start\_urls, we need to have a URL that our spider can begin crawling from, hence we call it the start URL, because this is the URL that the spider will begin crawling from.

\*whenever we crawl a web page, we need to have what we call the main page that we crawl, and this is what the start URL represents, it represents the main page that we need to crawl.

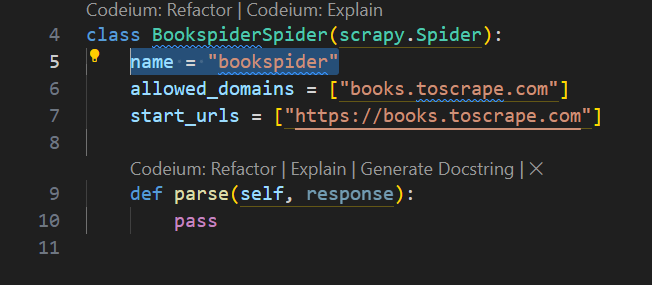
**Step 3**: Specify the allowed domains: (line 6)



\*this attribute simply specifies the domains that the spider is allowed to crawl through, this means that we specified the starting url, which is the main page that the spider needs to crawl, a situation can occur where the main web page is linked to other external web pages, and in this case we need to restrict the spider from crawling through other web pages, and this is where we restrict the spider, we don’t want the spider to crawl over the entire internet, so we need to ensure that we are able to restrict the spider. And this is where we restrict the spider, this is where we tell the spider which domains its allowed to crawl.

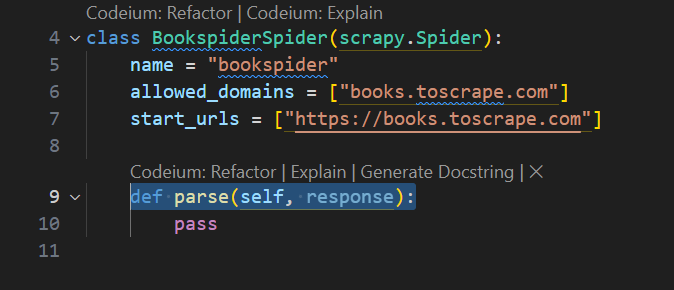
**Step 4:** The name attribute (line 5)

|  |
| --- |
| $ scrapy genspider bookspider books.toscrape.com  \* bookspider is the name of the spider  \* books.toscrape.com is the name of the website that we want to scrap |



\*the name attribute simply specifies the name of our spider, when we created the spider by issuing the commands given at the top, we named the spider “bookspider” and that’s what we are seeing here, its simply the name if the Spider that we created.

Step 5: parse() function (line 9)



\*we need to goi and define whats called a parse function, all what the parse function does is that it allows us to define the logic of how we want to parse the information that we read from the web page that’s we crawl.

\*what you need to keep in mind here is that the information will be stored in html tags, and each of the html tags will have css selectors that we can use to style the web pages. So when we crawl through the webpage we will use the html tags and we will use the css selectors that we find in the html tags

\*what we need to code is the logic of how we want the spider to crawl through the given web page, and how we want to the information that we find in the html tags to be extracted and stored, we can either store the information in a csv file, or we can store the information in a database, all of this logic is coded into the parse() function.

\*response parameter: when we look at the default parse() function that is created for the spider, one of the first things that we notice here is that we have a “response” parameter. To understand what that response parameter does we need to understand the following information here: the client sever model

The client-server model: this model simply describes how a web browser(client) is able to communicate with a web server and request a web page from the web server. What will happen is that the web-browser will send an http request to the web server, and the web server will send an http response back to the web-browser where it will respond with the given web page

-----------------------------------------------------------------------------------------------

**Scrapy Shell:**

\*once we have installed the scrapy shell using ipython, we need to run the scrapy shell



**List of scrapy shell commands:**

A screenshot of a computer program

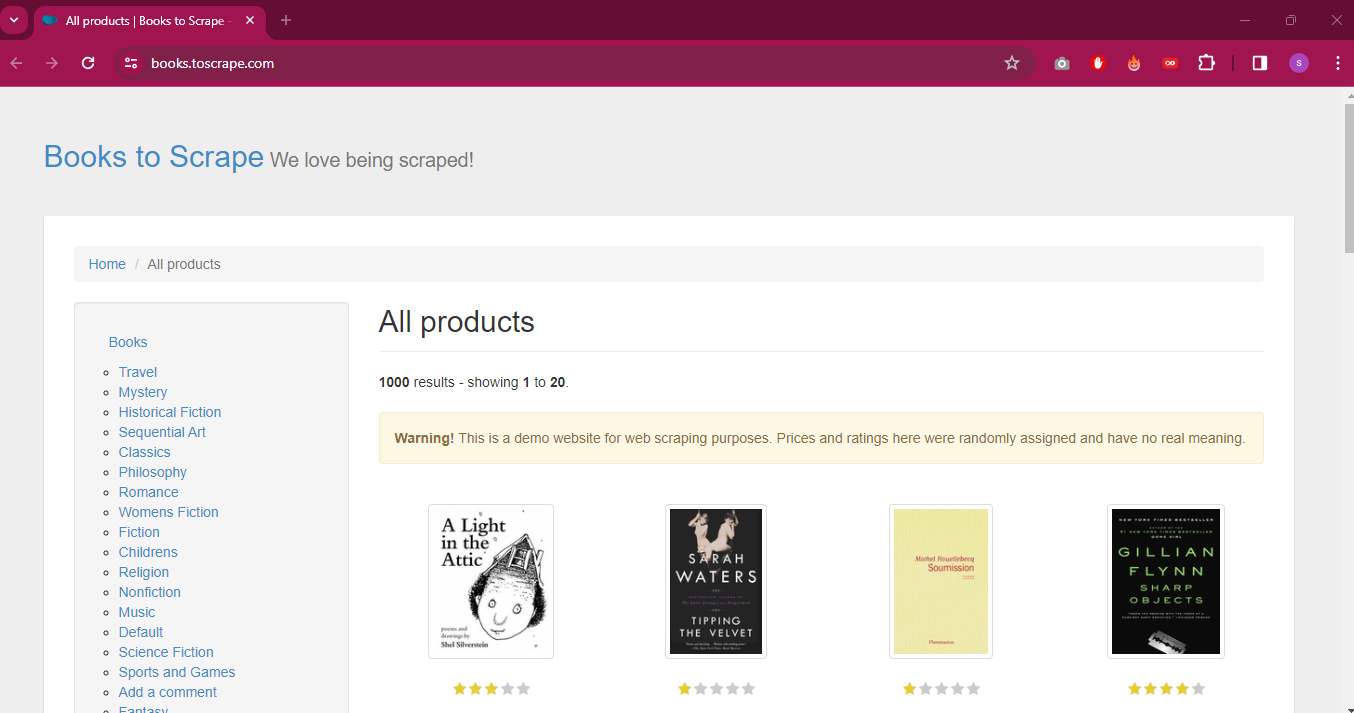
Description automatically generated

\*what we see here is that we have a list of all the scrapy shell commands that we can run. And the scrapy shell command that we are most interested in for now is the: fetch(url) command

**Step1** : we run the scrapy shell fetch(URL) command:

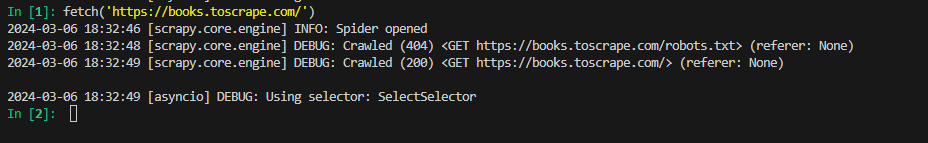
|  |
| --- |
| fetch('https://books.toscrape.com/') |

\*Remember that this is the website that we want to scrap data from: and this is the URL of the website : https://books.toscrape.com/



\*so what we need to do is to run the fetch command:

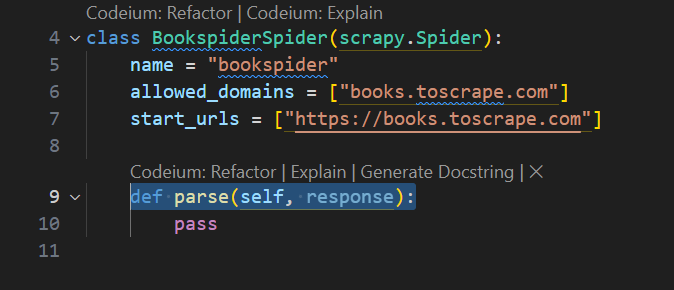
fetch('https://books.toscrape.com/')



\*so what the fetch command does is that it takes all the html + css of the webpage and it places it into this variable called response



This is why when we run this response command, we get back a 200 OK response, and this means that the server is ready to respond back to us with the web page that we need.



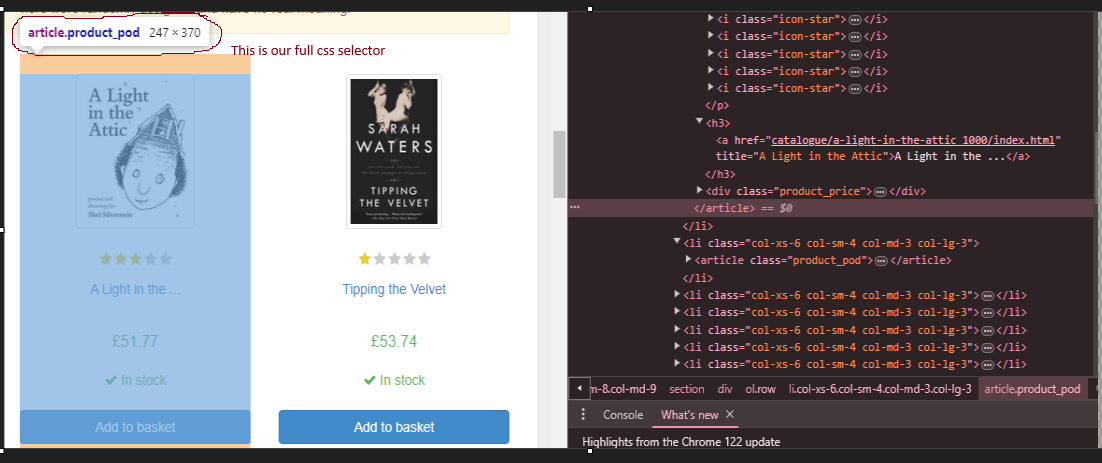
\*if you go back to the parse function, it had a parameter which was response, so all we have effectively done now is to take all the HTML + CSS and store it in this variable called response.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**How to manipulate the response variable:**

|  |
| --- |
| response.css(‘css\_selector’) |

\*we are going to use a function called: response.css() and we are going to have an argument in this function which is called the css selector



\*we can clearly see from the information that we have here that the whole image that we want is given by the following css selector = “article.product\_pod”, the entire shaded area and all the information in that shaded area is given by this css selector.

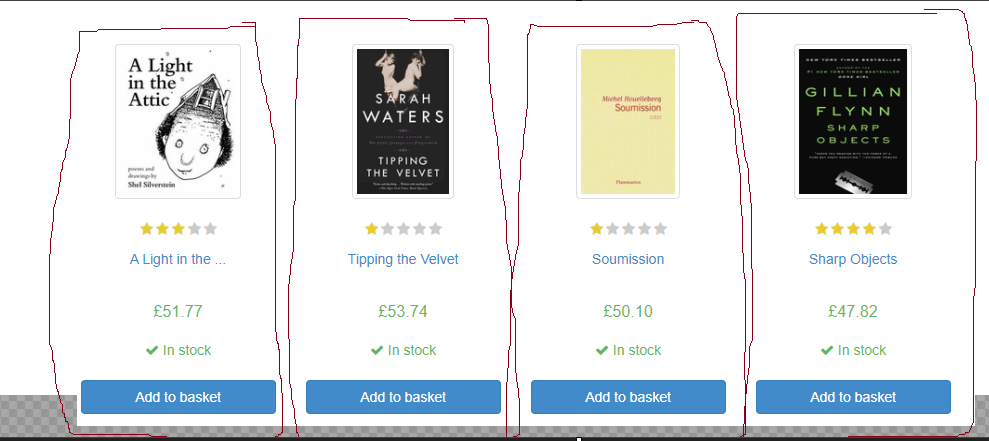
\*so now that we have identified the css selector that will give us the information that we need, the next step is that we need to use the function above which is response.css(‘css-selector’) and we need to provide the css selector as the argument in this function.

|  |
| --- |
| response.css('article.product\_pod') |



\*once we issue the command which is: response.css('article.product\_pod')

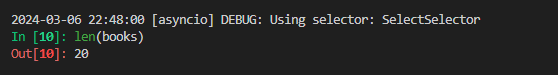
This is the response that we get, we get a list of all the areas that we marked on the page:



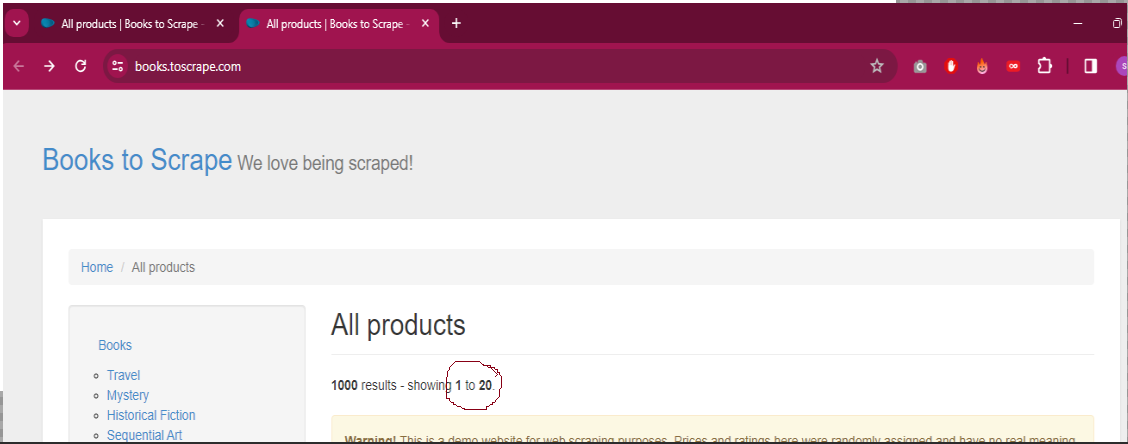
**How to store the data that we get from the response.css() query:**

\*The basic idea here is that we can create a python list variable that will allow us to store the response that we get using response.css(), so what happens here is that all of the data is stored in a python list. And because we have a python list it means that we can index the python list, we can iterate through the python list.





\*I can even run the len() function on books which will return how many fields I have stored in this python list. In this case I have 20 fields, because the page has a total of 20 books



\*So what we can do here next is that we can separate the books, and create a separate variable for each book



-----------------------------------------------------------------------------------------------

Okay for this part of the work we want to extract 3 things:

A screenshot of a book

Description automatically generated

1. The name of the book



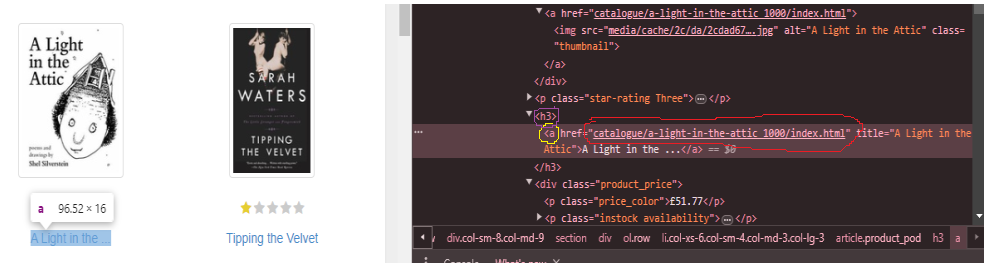
|  |
| --- |
| .css(‘h3 a::text’).get() |

\*I need to go and specify the data type of what I need this selector to return, and in this case I want it to return “text” and I also have to use the get() method at the end.

\*also notice that im using html tag elements and I don’t have a class selector,

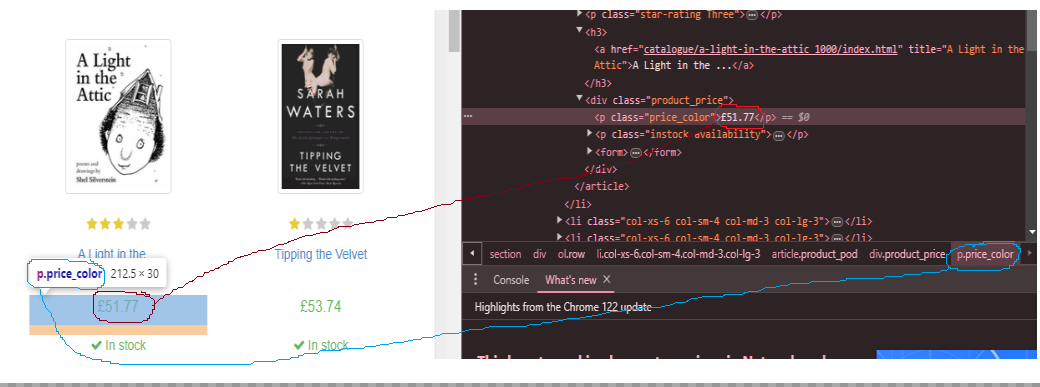
1. The url of the book

|  |
| --- |
| .css(‘h3 a’).attrib[‘href’] |



1. The price of the book

|  |
| --- |
| .css(‘p.price\_color::text’).get() |

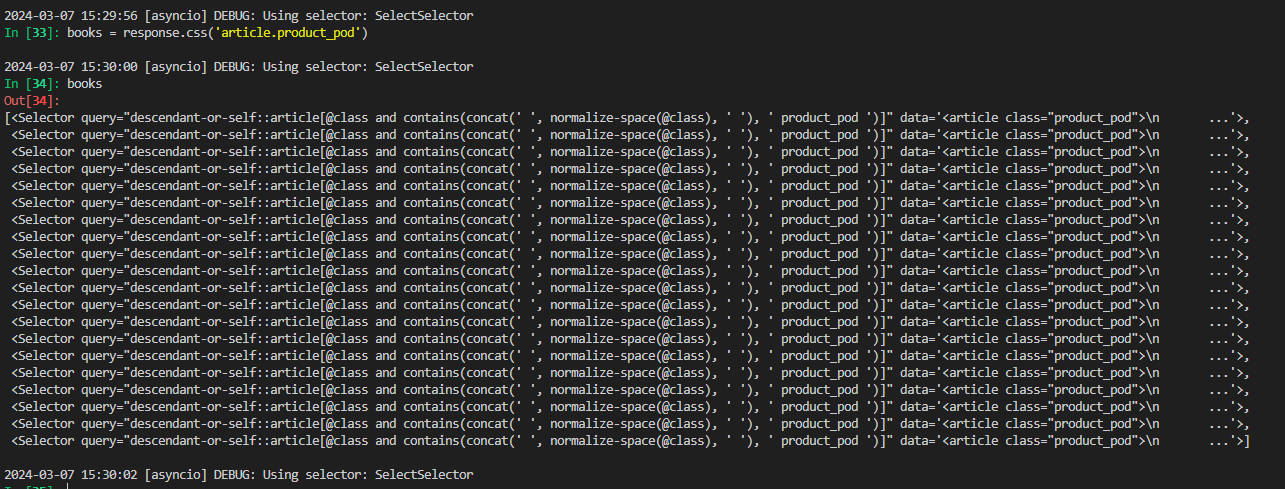


\*but we need to know the css selectors for this information, and that’s the first thing that we need to get.

------------------------------------------------------------------------------------------------

**How do we extract all the given data into our parse() function:**

**Step1**: we need to go and create a variable that’s able to extract all the given information about the books.



|  |
| --- |
| Books = response.css('article.product\_pod') |

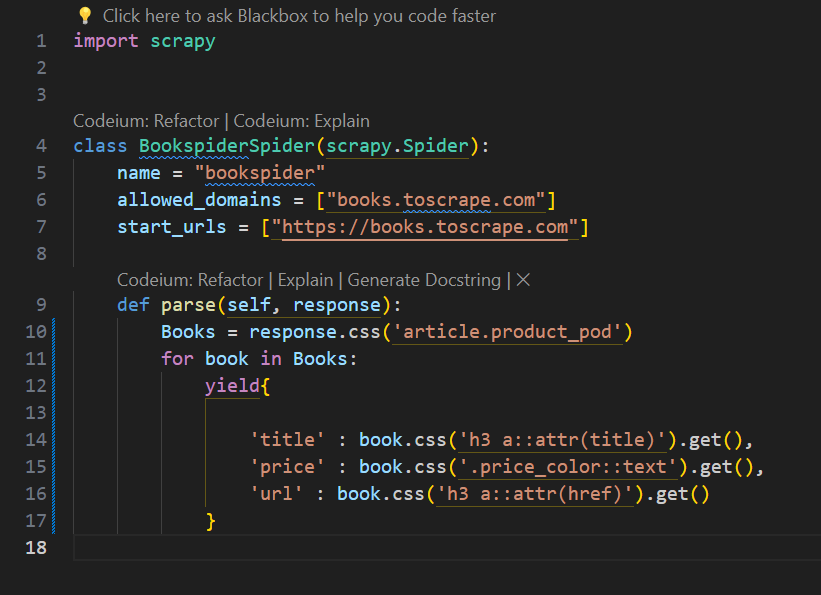
**Step 2**: we need to create a for loop that’s able to iterate through all of the books, and return(yield) all of the information relating to a book

|  |
| --- |
| for book in books:  yields{  }  \*In this function, what happens is that each information about a book, will be stored in the variable called book, and what I need to go and specify is the format that I want the information to be extracted in.  \*I want the:  ‘name’ :  ‘title’ :  ‘url’ : |

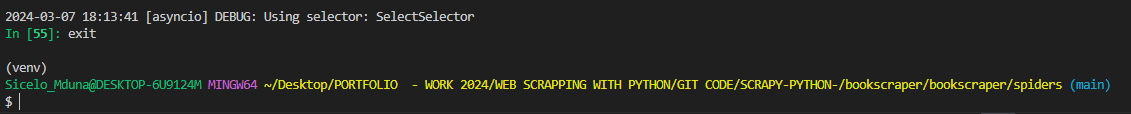
\*so the basic idea here is that we always use the get() function, whether we want to return an attribute or simple text.

------------------------------------------------------------------------------------------------

How do we run the spider and see the results:



**Step 1**: we need to exit the scrapy shell



**Step 2**: we need to navigate back to the ‘bookscraper” folder

A black rectangle with yellow text

Description automatically generated

**Step 3**: we need to run the scrapy crawl and indicate the name of the spider which is “bookspider”



**Results:**

\*The results will show all the information that we asked the parse() function to yield, which is the:

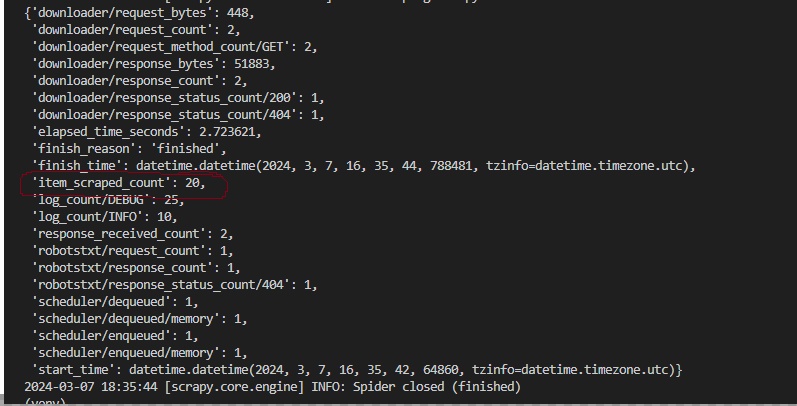
‘title’:

‘price’:

‘url’

\*it will return these results for all of the books on the page which is 25 books in total, because we used a for loop



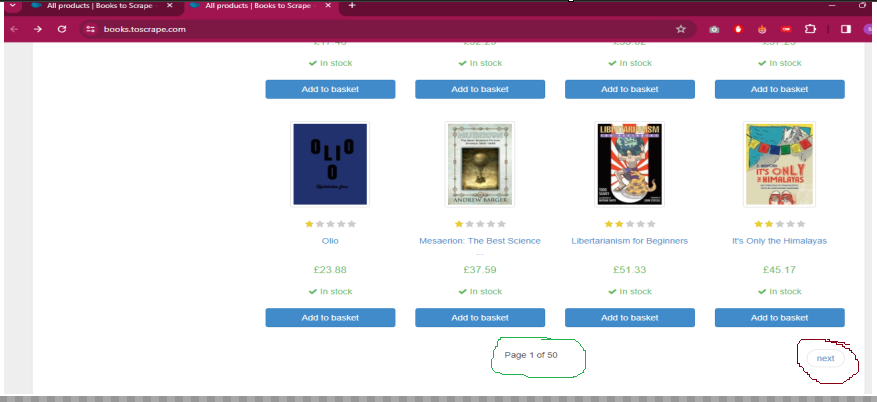
\*item\_scraped\_count = 20, indicated how many results were returned when we ran the program, and in this case this returns 20 results, because we have 20 books on the page.

###############################################################

**PART2: HOW DO WE SCRAP THROUGH MULTIPLE PAGES:**

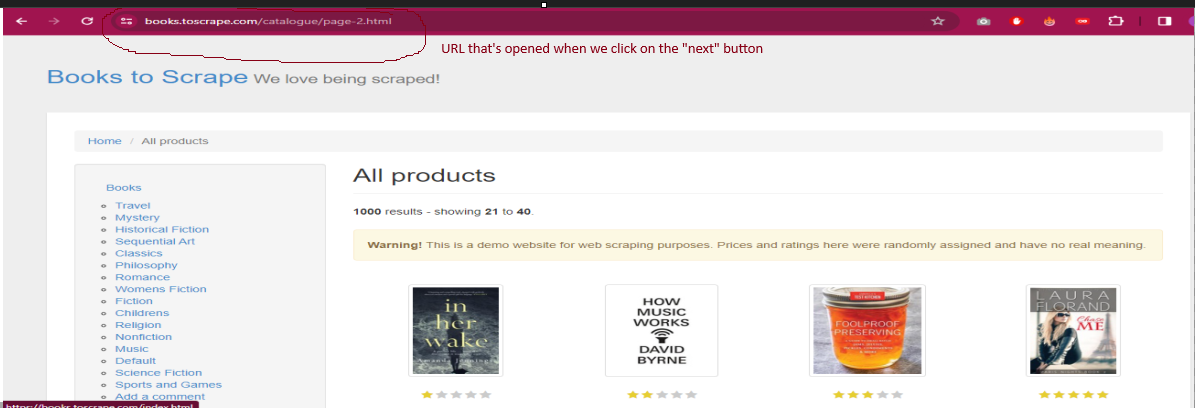
|  |
| --- |
| The next part of our tutorial focuses on how we can scrap through multiple pages, and how we can easily create a loop that will scrap through all of the pages until we have no more pages that are left |

\*As we can see from the information below, we have scraped data from page 1 of 50, this means that we have a total of 50 pages that we need to scrap data from in total and we have only scraped data from 1 page. We can also see from the picture below that we have a “next” button that we can click on in order to see content on the next page.



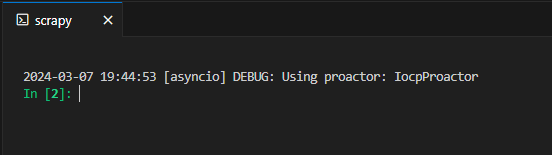
\*when we click on the “next’ button, the URL opened is:

|  |
| --- |
| https://books.toscrape.com/catalogue/page-2.html |

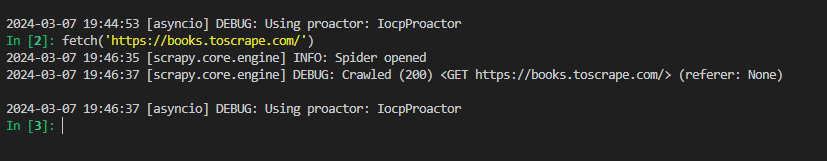


**Step1**: we need to launch “scrapy shell” inside of our gitbash command



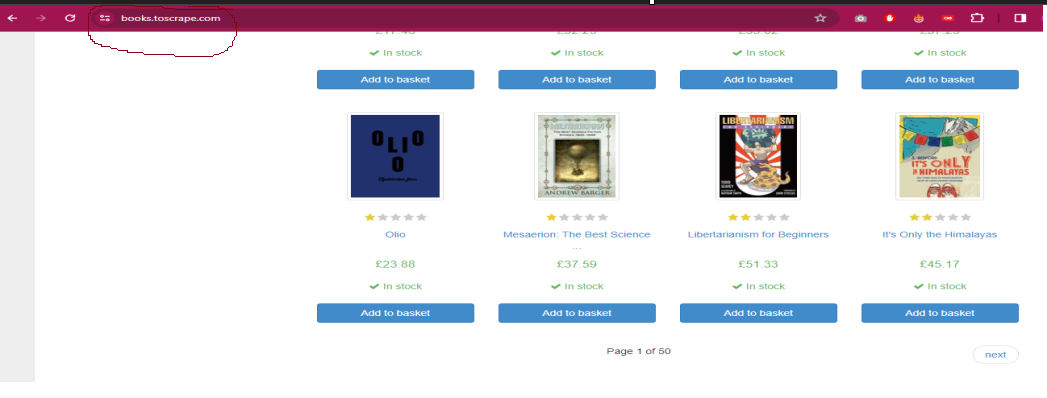


**Step 2**: we need to run the fetch command with the url of the website

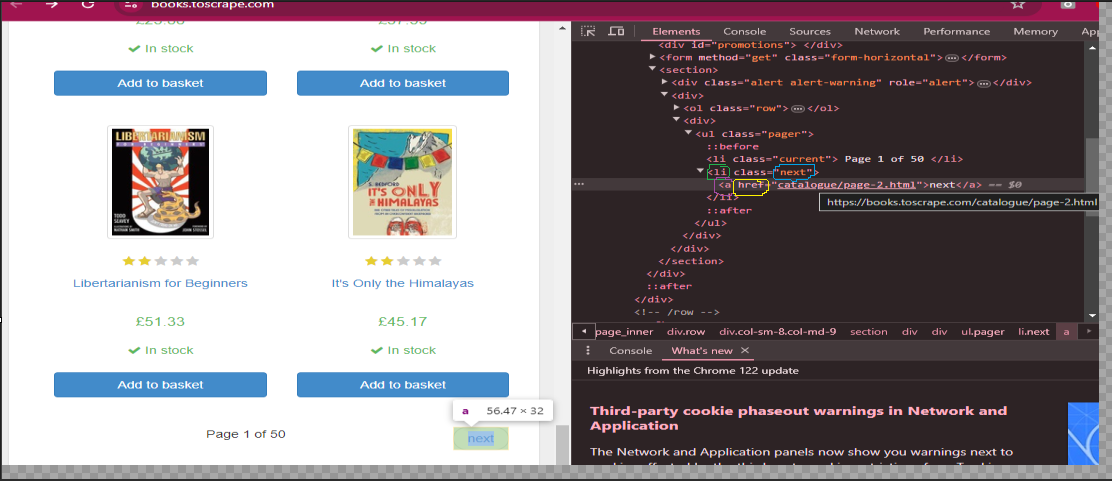


**Step 3**: we need to go and get the link that takes us to the next page

***Step 3.1***: we need to open the webpage that we fetched in any web browser and in my case the web-browser im using is google chrome:



***Step 3.2 :*** I need to inspect the page, in order to get the get css selector that takes me to the next page(which is a url)



The css selector that take me to the next page is:

|  |
| --- |
| response.css(‘li.next a::attr(href)’).get() |

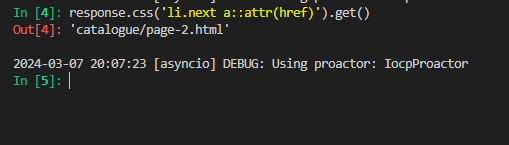
*Step 3.3*: we need to test out this css selector within scrapy shell and see if it returns the result that we want, we need to go and create a response query, that is able to extract all the information that we need from the given page

A screenshot of a computer

Description automatically generated

The css selector that take me to the next page is:

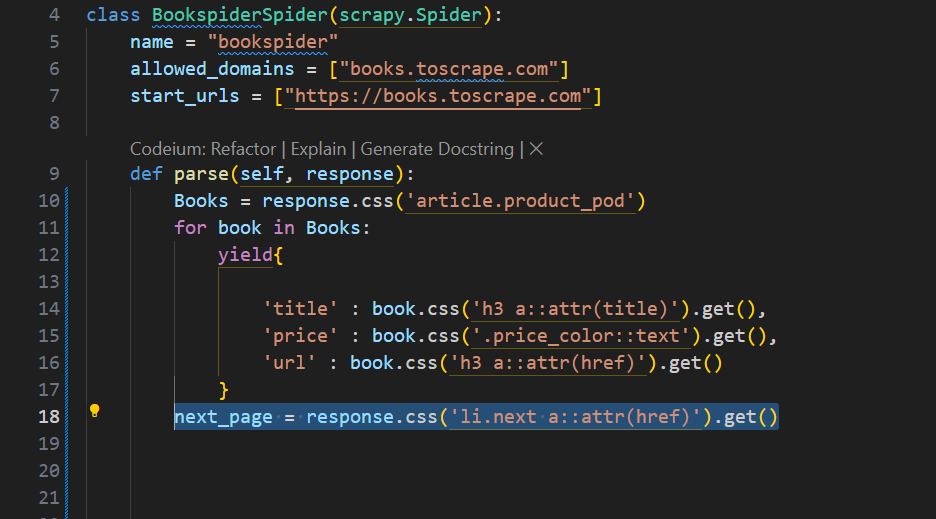
|  |
| --- |
| response.css(‘li.next a::attr(href)’).get() |



\*we can see here that when we give the scrapy shell command and we pass the css argument into it, we get back the response that we expect which is the url that takes us back to the next page

**Step 4:** We now need to go and code the logic that allows us to loop through all of the pages(from page 1), until we reach the last page(page 50)

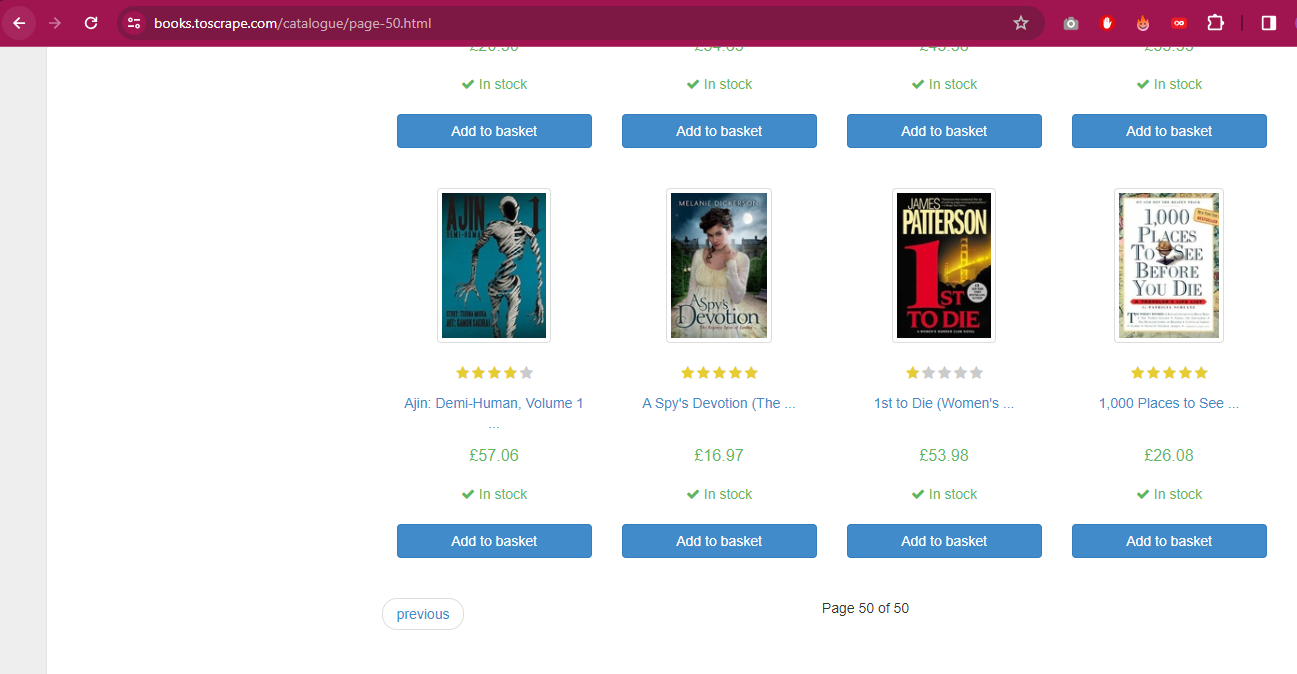
\*The very first thing that I need to do here, is that I need to create this variable called “next\_page” that allows me to retrieve the url of the next page

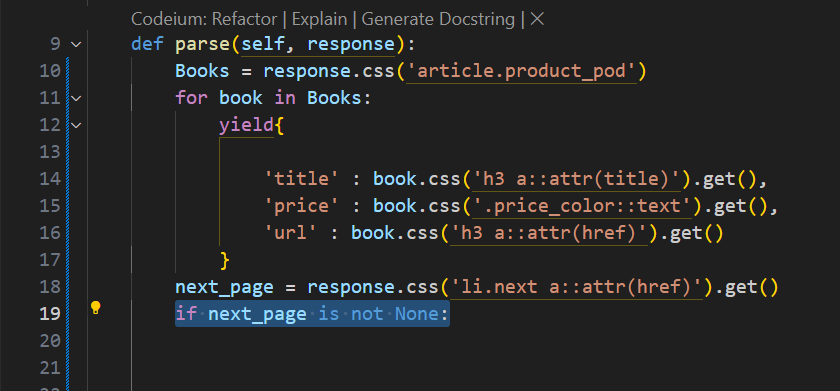


\*so what I need to go and do next is that I need to write an if, conditional statement that is able to check if I have reached the end of the page.

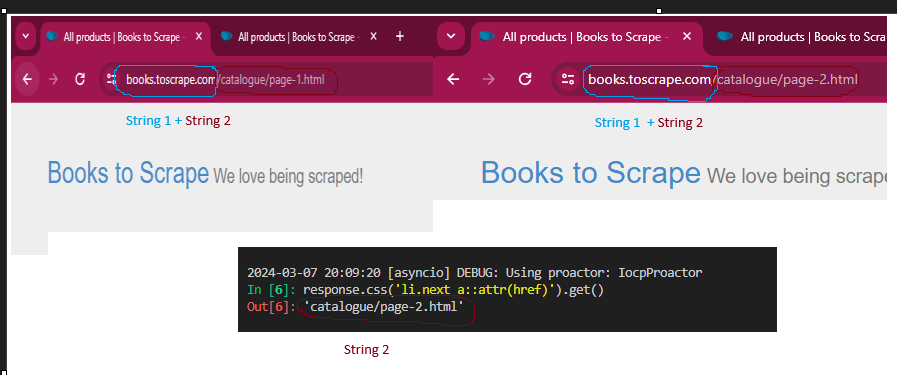
\*what you need to take note of here, is that when I reach the end of the pages, which in this case is page 50 of 50, what notice here is that I no longer have the “next” button, and this is the indication that I have reached the end of the pages, and that’s what I will use in order to set up the if conditional statement that tells me that I have reached the end of the page.

\*this means that every other page has the “next” url, expect for the last page, and how I code that in python, I need to create an if conditional statement, that says that “if next\_page is None:”, this keyword “None” allows me to specify that I have reached the end of the pages. It means that the “next\_page” url has disappeared.





Now I need to code the logic that allows me to scrape into the next page, this if conditional statement does the following, it says that if the “next\_page is None” this means that if I have not reached the end of the page, this is what I need you to do.



\*what we need to notice here is that next\_page returns String 2, so that’s what we have stored In this variable called next\_page we have String\_2

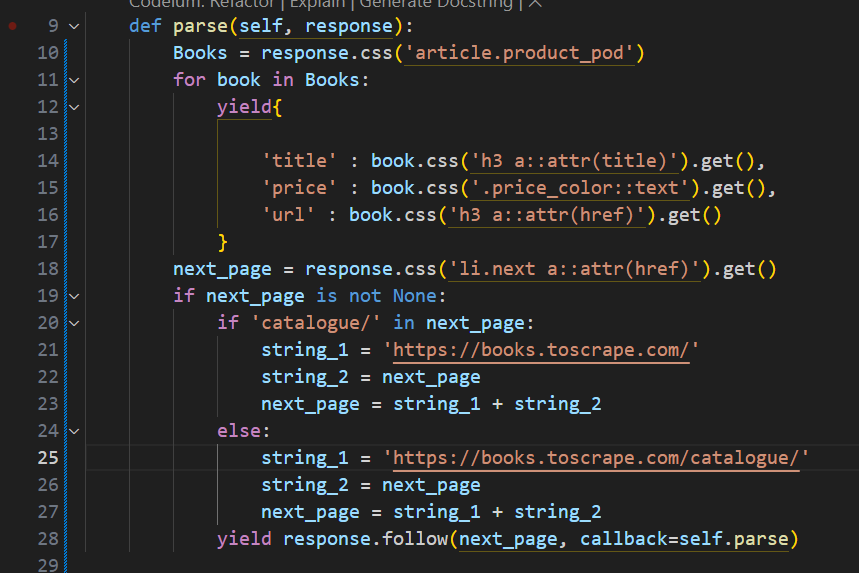
\*if we look at the url of each page, we can clearly see that the url is split up into 2 parts;

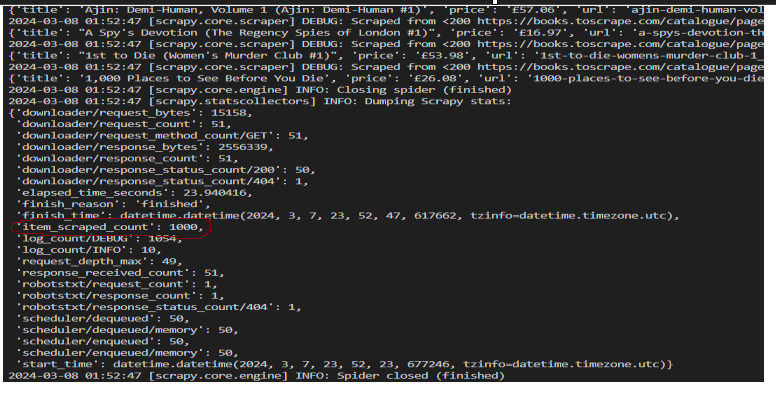
String\_1 = <https://books.toscrape.com/>

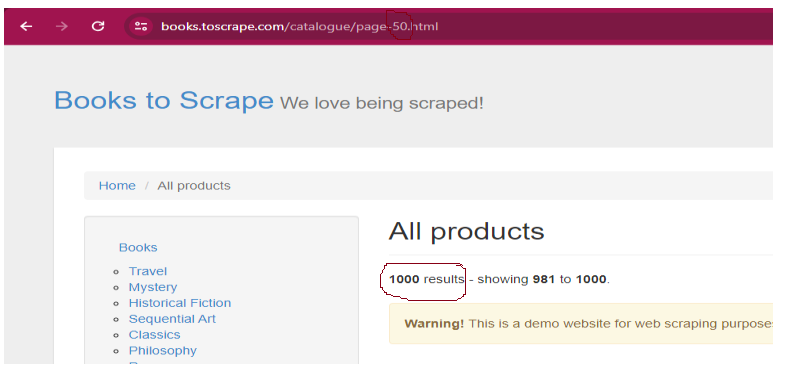
String\_2 = next\_page

\*this means that the next page, will always be a string concatenation of String\_1 and String\_2

next\_page = String\_1 + String\_2







\*The item scraped count is: 1000, which corresponds to the total search result which is 1000

------------------------------------------------------------------------------------------------