**How to Set up a Virtual Environment in python:**

The very first thing that we need to do is that we need to set up a virtual environment

Step1: Create a folder that you want to work in

A screenshot of a computer

Description automatically generated

**Step 2:**

Open this folder in gitbash

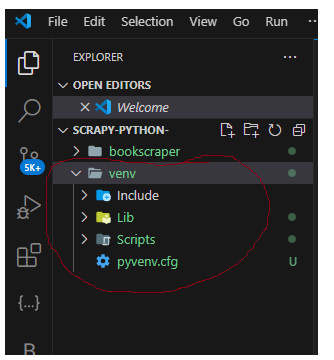


**Step 3:**

Type the following command: *python -m venv venv*

Where venv, the second argument is the name of the virtual environment that we want to create.

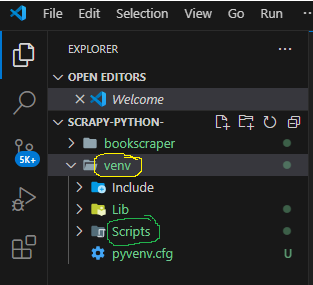
This will create the following folder:



**Step 4**: activate the virtual environment

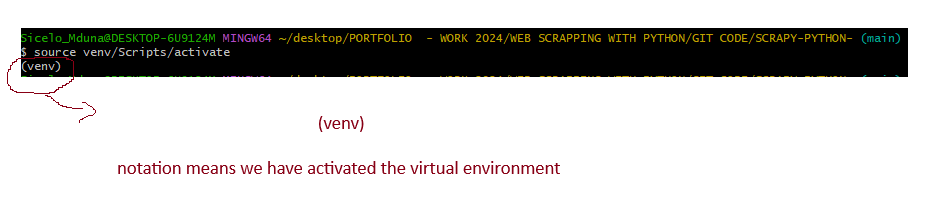
Type in the following: source venv/Scripts/activate

Where venv/Scripts



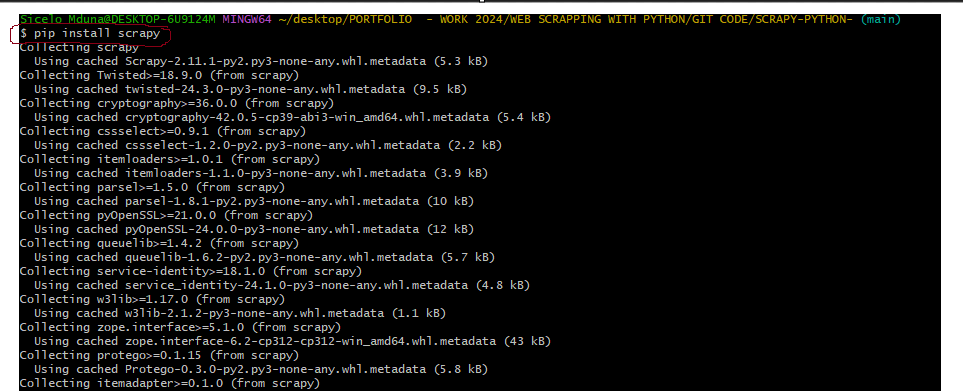
After we type this command in gitbash we should see the following notation which indicates that we have activated the virtual environment:

(venv)



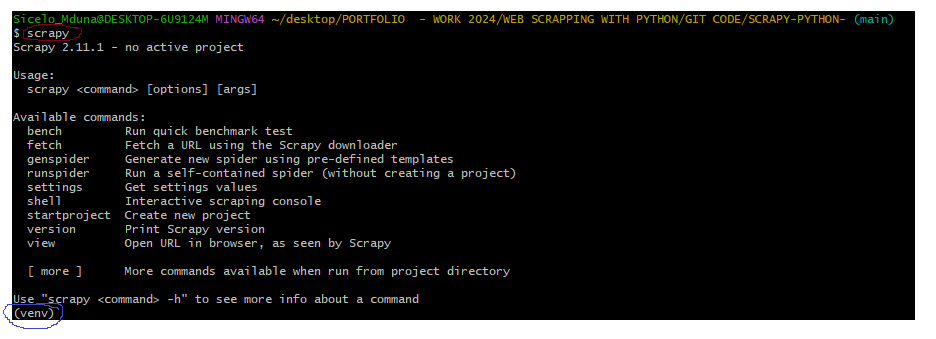
**Step 5**: we need to download Scrapy into the virtual environment that we have created and activated:

Type: *pip install scrapy*



We can use the command : *scrapy*

To see if scrapy has been installed and this is the screen that should appear in gitbash to confirm that scrapy has been successfully installed:



\*again notice the (venv) at the bottom of the screen which is highlighted in red, this confirms that we have a virtual environment that has been activated, and the module that we installed is installed on top of that virtual environment, which is what we want, hence we created the virtual environment in the first place.

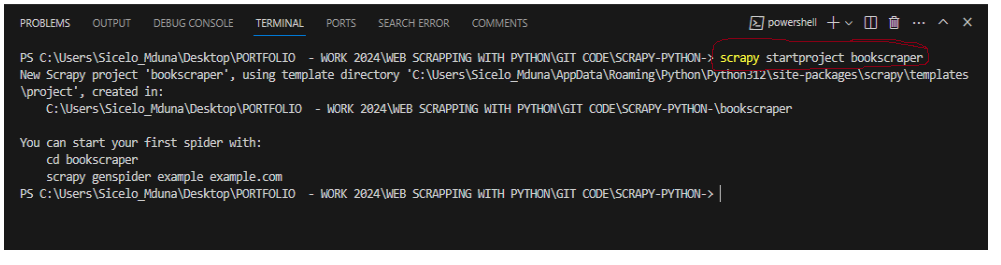
**Step 6**: we need to create a new Scrapy Project

\*to create a new Scrapy project we can do this directly in visual studio and this is the command that we have to type in a new terminal in visual studio code

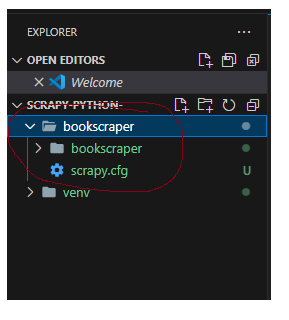
Type: *scrapy startproject bookscraper*

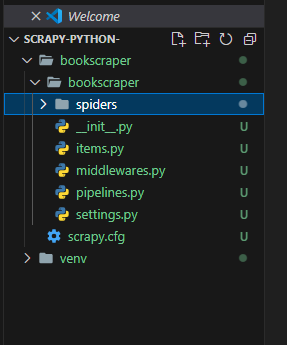
Where *bookscraper* is the name of the Scrapy project

\*Once we have successfully created the Scrapy project in visual studio code, we should see the following message, which confirms that we have done the right thing and the project has been successfully crated:



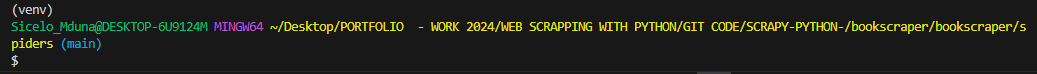
This will and create a new folder in visual studio that looks like this;



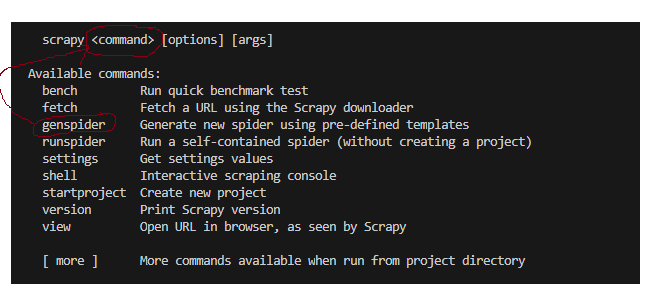


\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Step 1**: we need to navigate to the “spider folder” using the gitbash terminal in visual studio



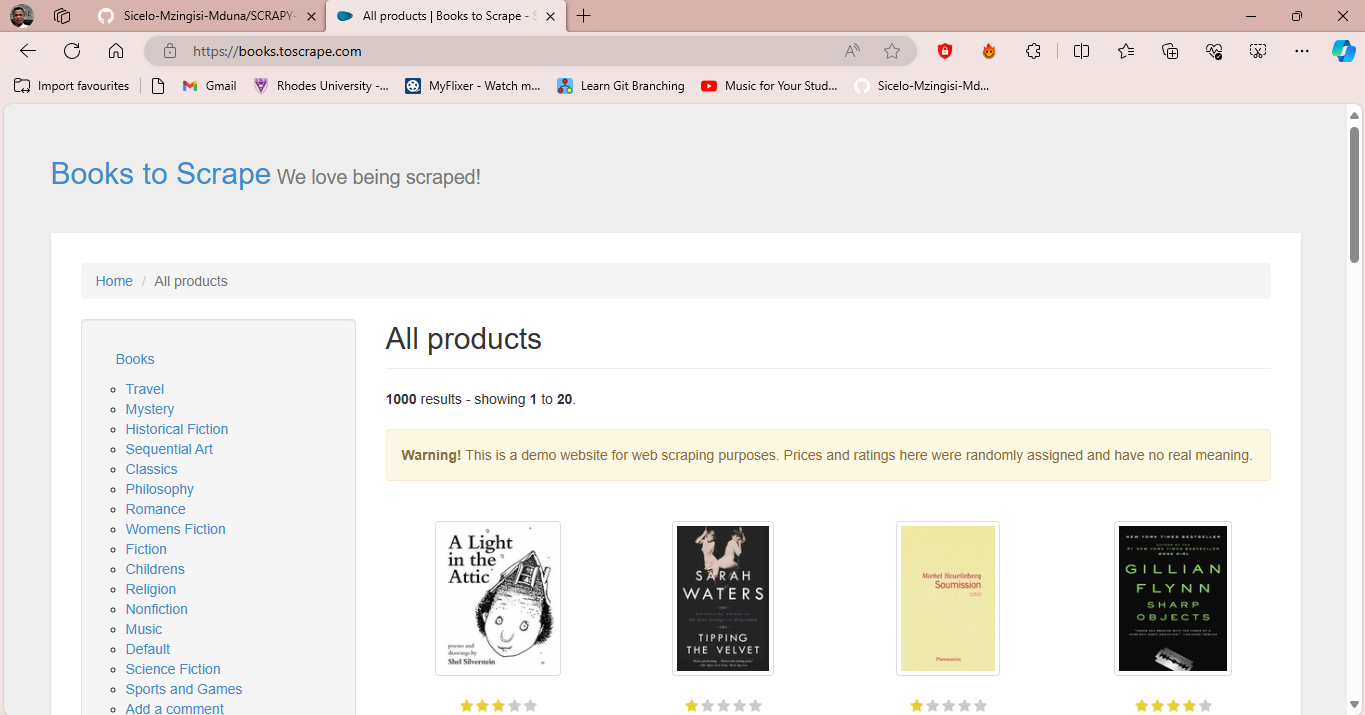
**Step2**: we need to run the following gitbash command



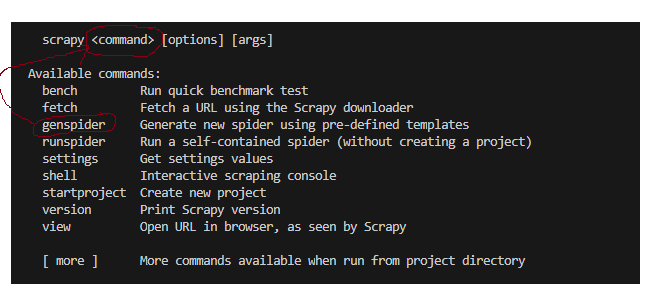
$ scrapy genspider bookspider books.toscrape.com

\* bookspider is the name of the spider

\* books.toscrape.com is the name of the website that we want to scrap

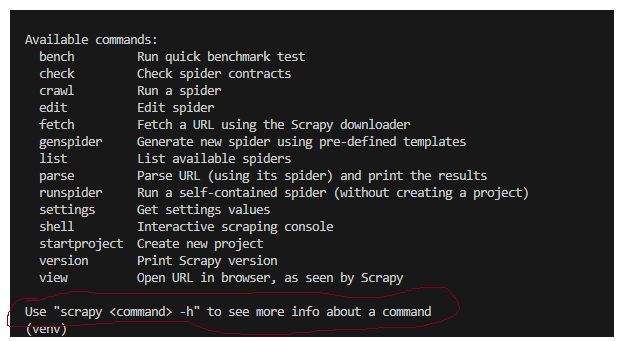


As we can see here this is a demo website that has been set up, for the simple purposes of allowing us to scrap data from it, it allows us to simply practice our skills.



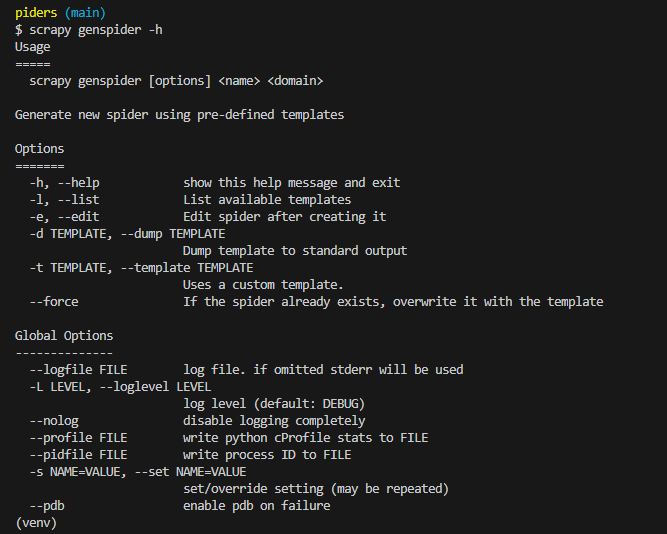
\*so genspider is a command that allows us to generate a new spider using a pre-defined template

\*we can find out more information about a command and how that command works using the following information:

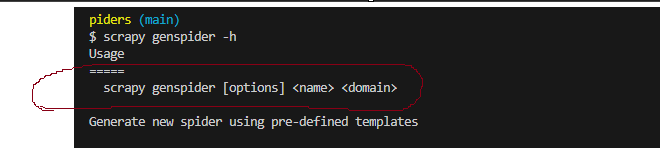


This gives us the whole format of what we need to do, we need to type in the following command in order to find out more information about a given command, in this case we want to find out more information about the “genspider” command so this is what we need to type:

|  |
| --- |
| scrapy genspider -h |



Information 1:

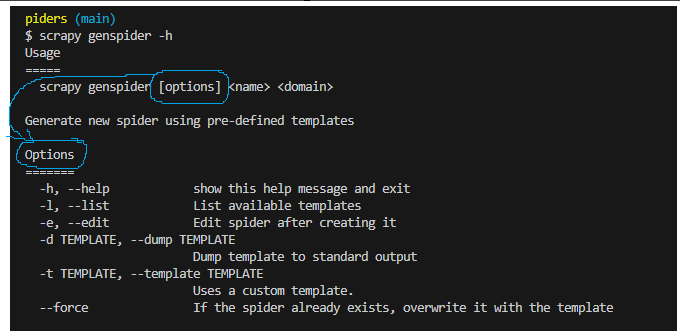


\*the first thing that we are told here is how we have to use the scrapy genspider command, and the respective arguments that the command goes with. So this is how the command goes and this is what we need to understand here.

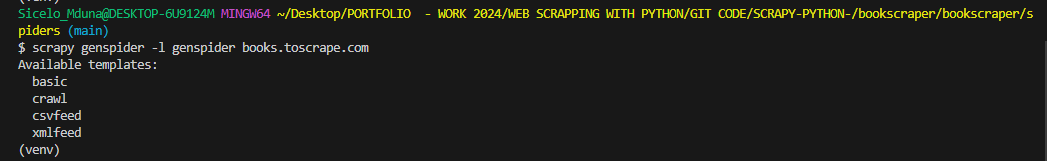
|  |
| --- |
| *scrapy genspider [option] <name> <domain>* |

\* <name> is the name of our spider, whatever name we want to give to our spider, this is the name that we give it. It can be anything that we want, and in this case the name of our spider is <bookspider>

\* <domain> is the domain name of the website that we want to crawl using the spider, and in this case the domain name of the site that we want to crawl is “books.toscrape.com”



|  |
| --- |
| scrapy genspider -l genspider books.toscrape.com |



\*the command that we used here is [-l] which will list all the available templates that we can use. And we can see here that we have 4 available templates. None of these arguments are ever supposed to have any types of brackets when we specify them whether angle brackets or anything of that kind.

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

Ipython Shell:

Step1: we need to install the ipython shell



*pip install ipython*

Step 2: we need to configure the ipython shell

A screen shot of a computer program

Description automatically generated

Step 3: we need to run the ipython scrapy shell



**List of scrapy shell commands:**

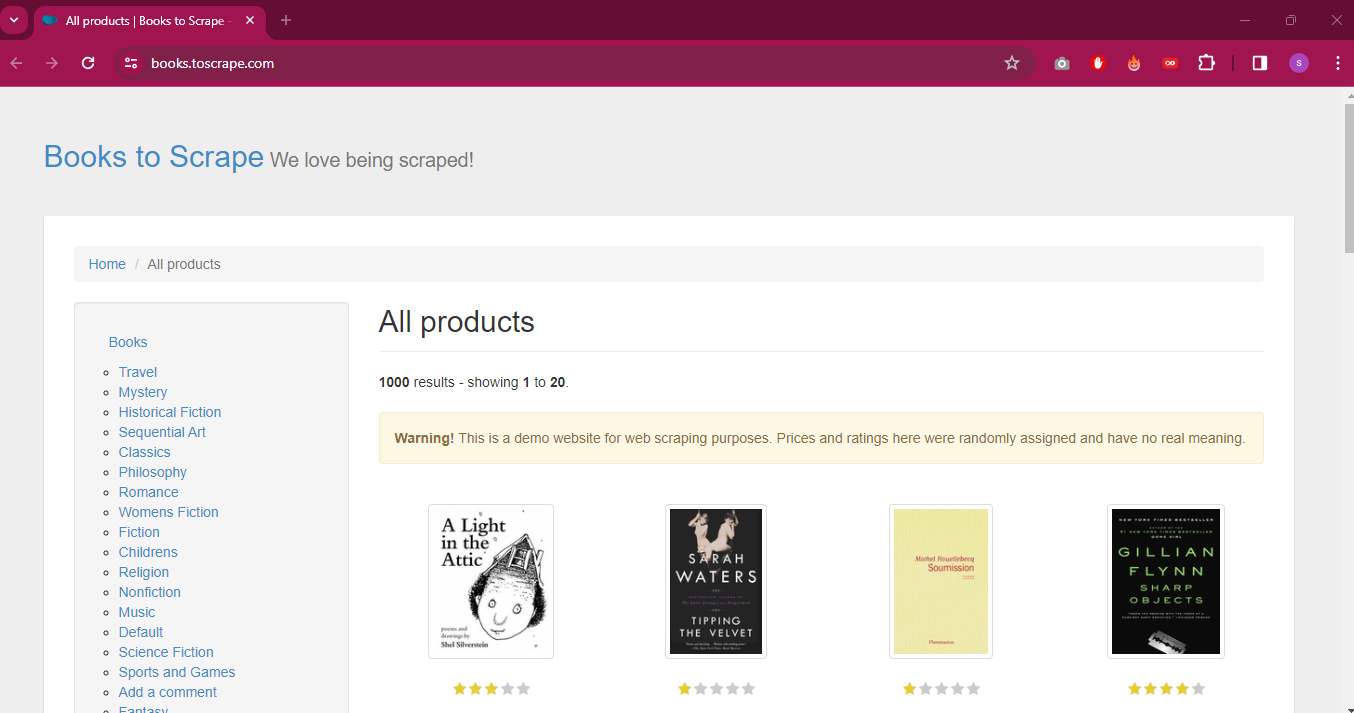
A screenshot of a computer program

Description automatically generated

\*from this list of scrapy commands, what we need to do is to run the: fetch(URL) command

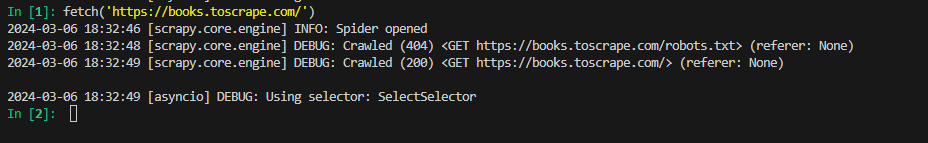
**Step1** : we run the scrapy shell fetch(URL) command:

\*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.

A screen shot of a computer program

Description automatically generated

\*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.

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

**Response commands:**

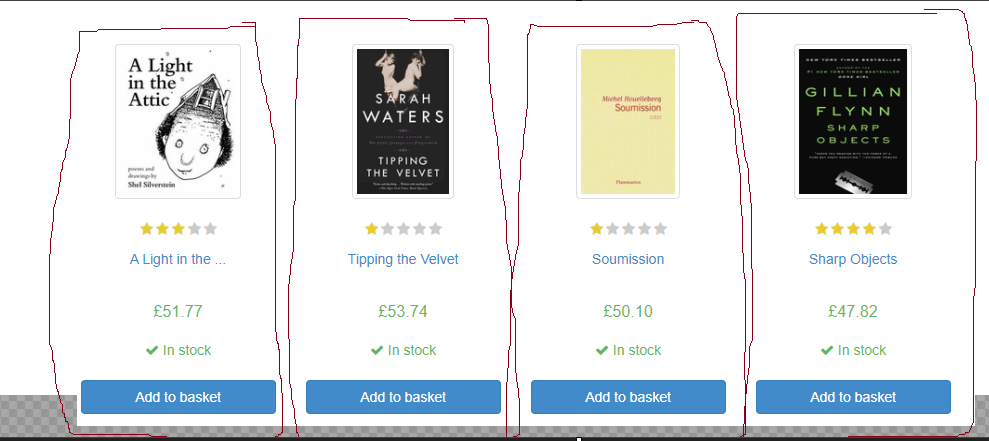
**Command number 1:**

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



\*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:



1. The name of the book

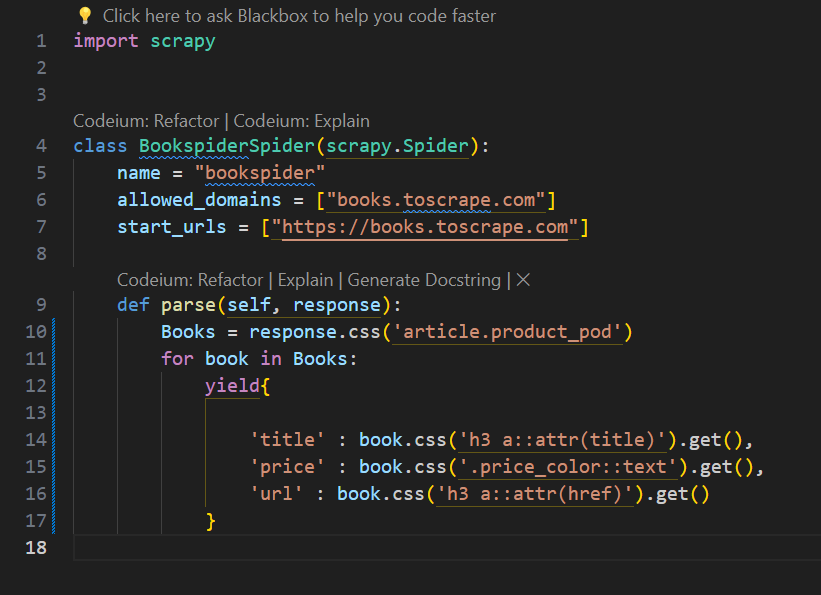


|  |
| --- |
| .css(‘h3 a::text’).get() |

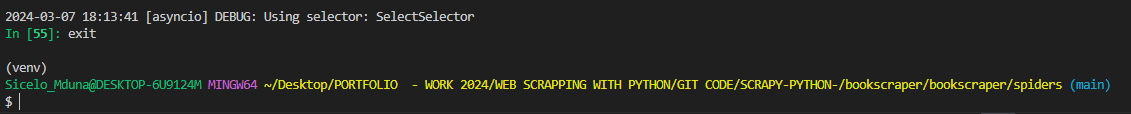
\*I need to go and specify the data type of what I need this selector to return, and in this case I want it to return “text” and I also have to use the get() method at the end.

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

**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’:

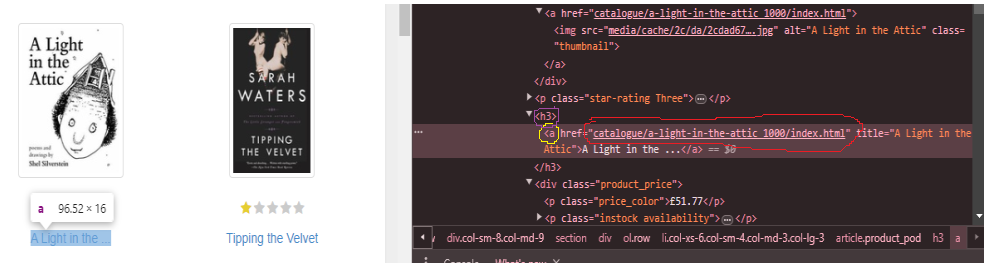
‘url’

\*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



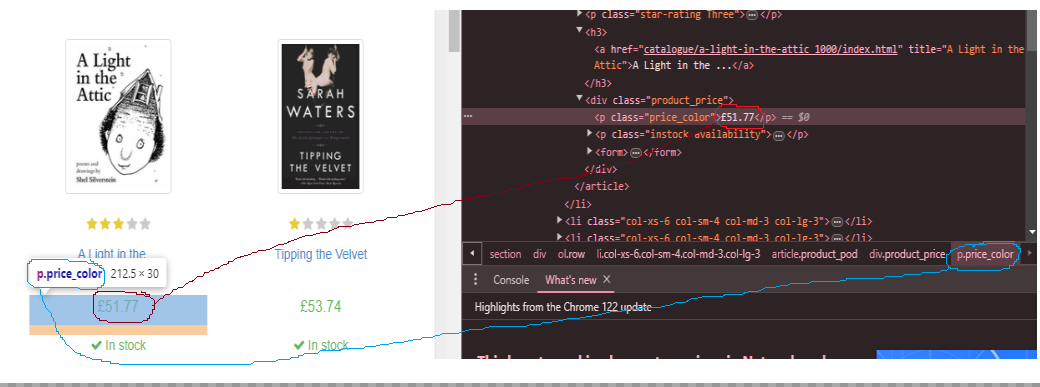
1. The url of the book

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

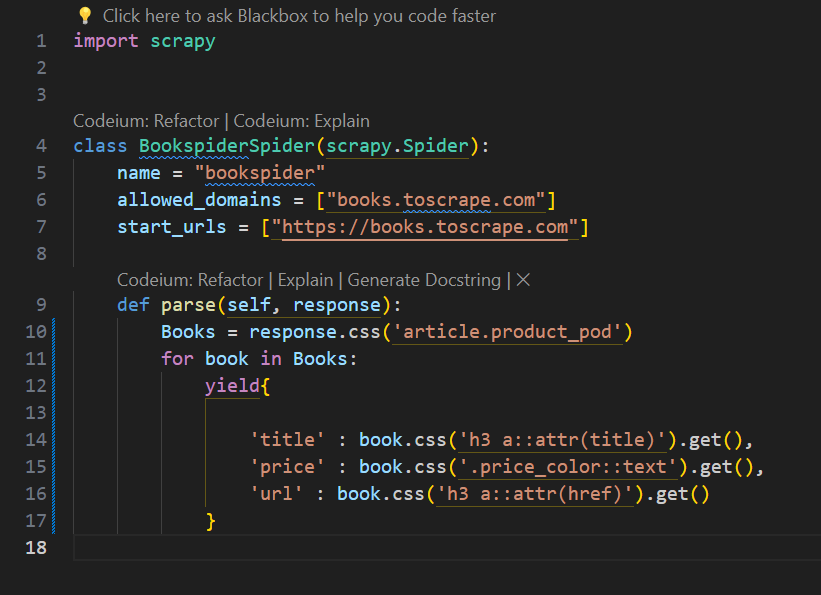


1. The price of the book

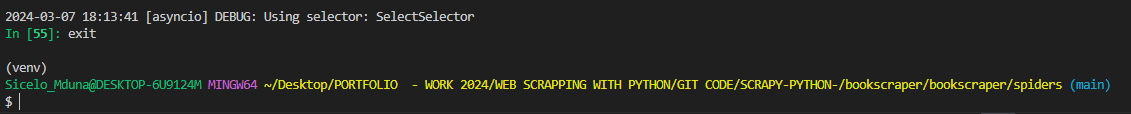
|  |
| --- |
| .css(‘p.price\_color::text’).get() |



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



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

**How to scrape through Multiple Pages of data**

**Step1:** we need to relaunch “scrapy shell” inside of our git bash command



A screen shot of a computer

Description automatically generated

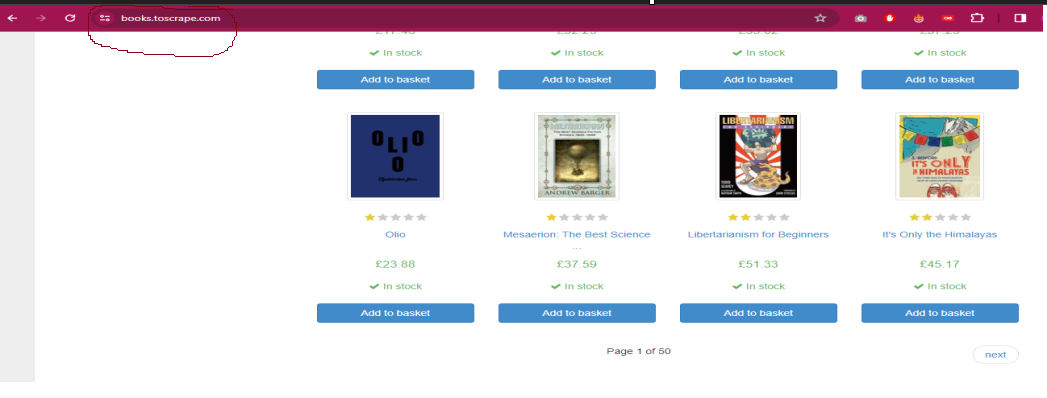
**Step 2**: we need to run the fetch command with the url of the website

A screenshot of a computer program

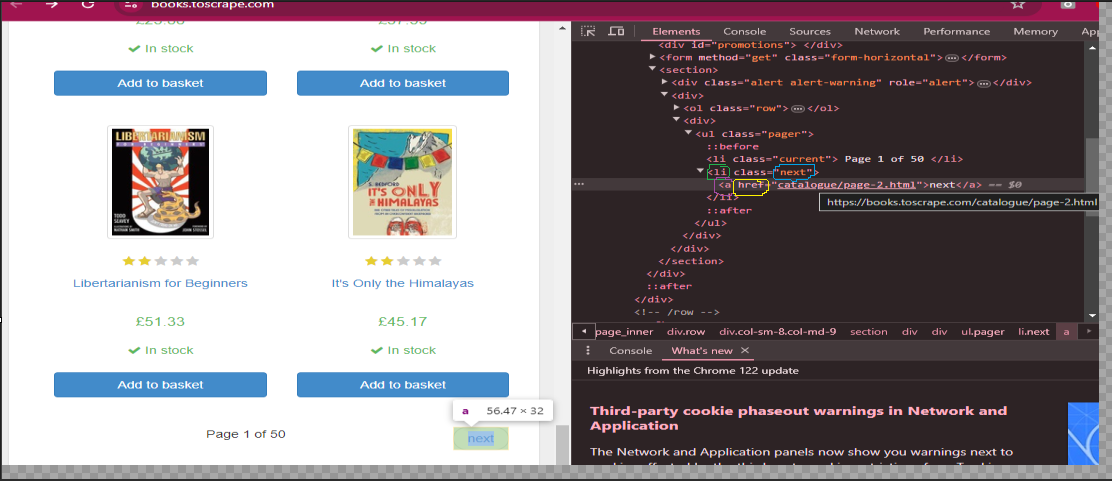
Description automatically generated

**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() |

A screen shot of a computer program

Description automatically generated

\*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

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