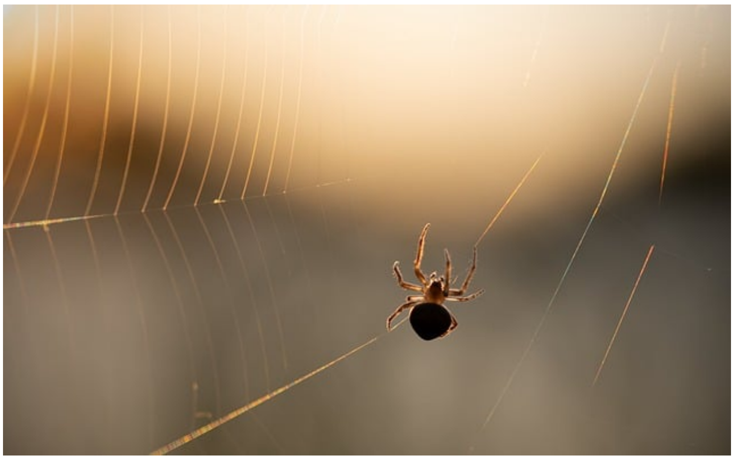
Topic: Web Scraping 27: Setup & Test Applications (WebScraper & Wikipedia)

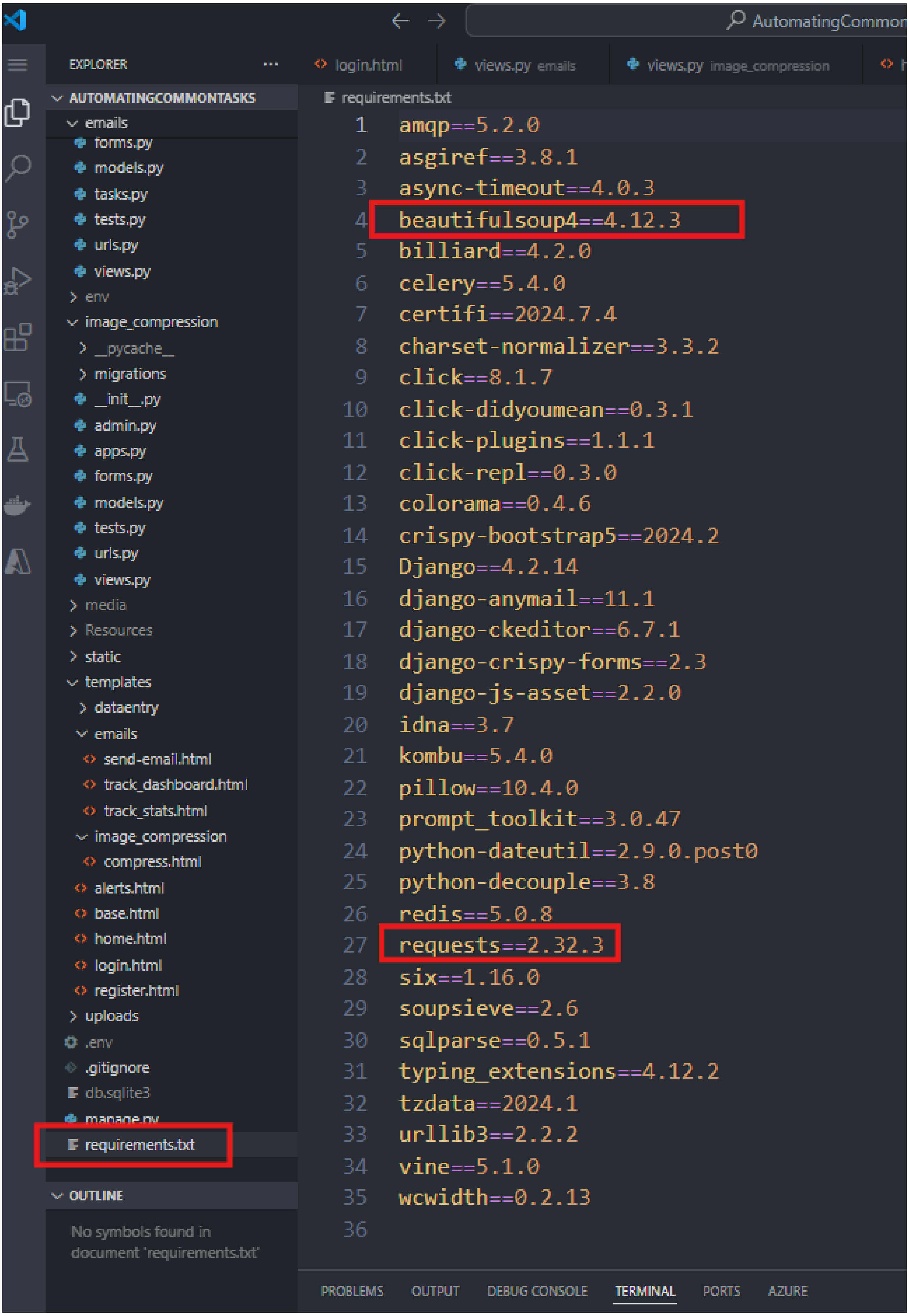
Speaker: | Notebook: Django: Automating Common Tasks



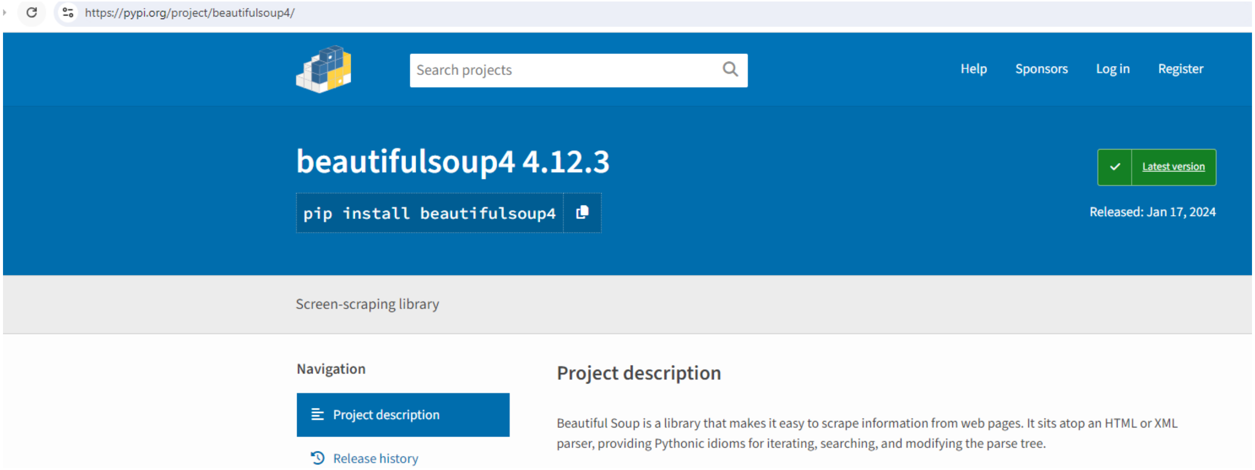
Web scraping is an automatic method to obtain large amounts of data from websites (see full documentation here)

1. To start, we need the libraries BEAUTIFULSOUP (which we previously download) and REQUEST Django library. Install these packages if these are NOT present in your REQUIREMENTS.TXT

```
$ pip install requests
```



```
amqp==5.2.0
asgiref==3.8.1
async-timeout==4.0.3
beautifulsoup4==4.12.3
billiard==4.2.0
celery==5.4.0
certifi==2024.7.4
charset-normalizer==3.3.2
click==8.1.7
click-didyoumean==0.3.1
click-plugins==1.1.1
click-repl==0.3.0
colorama==0.4.6
crispy-bootstrap5==2024.2
Django==4.2.14
django-anymail==11.1
django-ckeditor==6.7.1
django-crispy-forms==2.3
django-js-asset==2.2.0
idna==3.7
kombu==5.4.0
pillow==10.4.0
prompt_toolkit==3.0.47
python-dateutil==2.9.0.post0
python-decouple==3.8
redis==5.0.8
requests==2.32.3
six==1.16.0
soupsieve==2.6
sqlparse==0.5.1
typing_extensions==4.12.2
tzdata==2024.1
urllib3==2.2.2
vine==5.1.0
wcwidth==0.2.13

```



beautifulsoup4 4.12.3

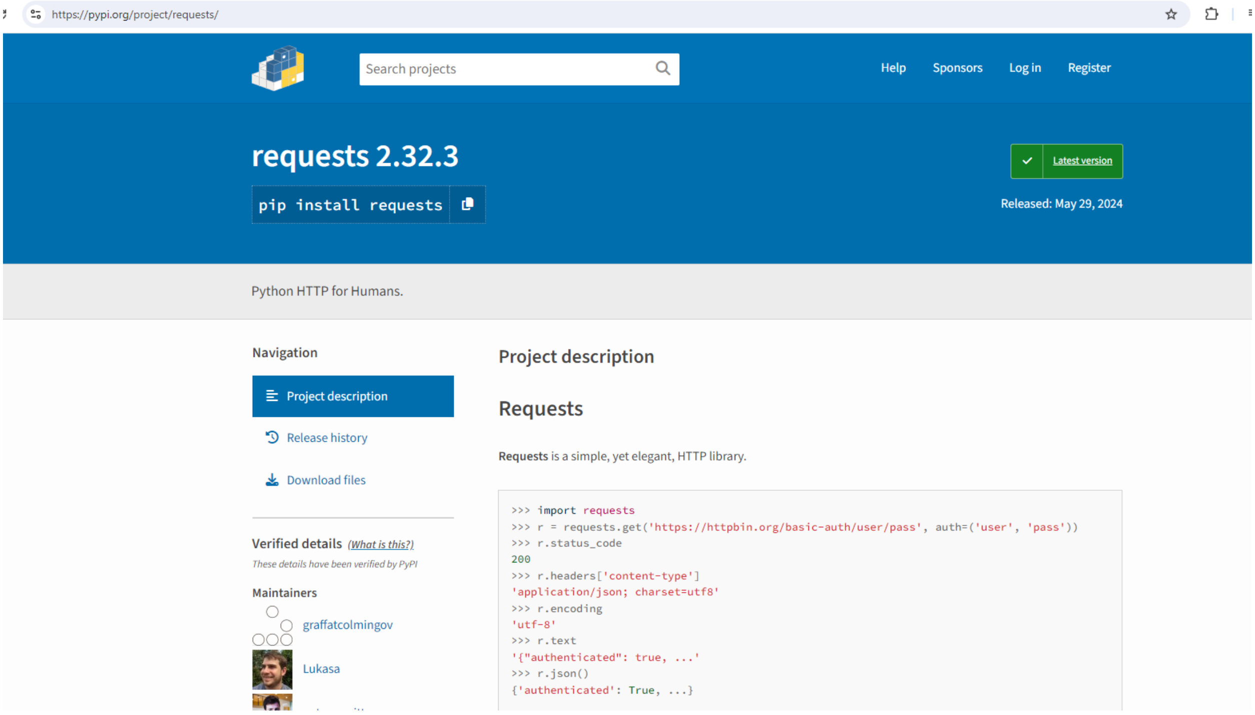
```
pip install beautifulsoup4
```

Released: Jan 17, 2024

Screen-scraping library

Project description

Beautiful Soup is a library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree.



requests 2.32.3

```
pip install requests
```

Released: May 29, 2024

Python HTTP for Humans.

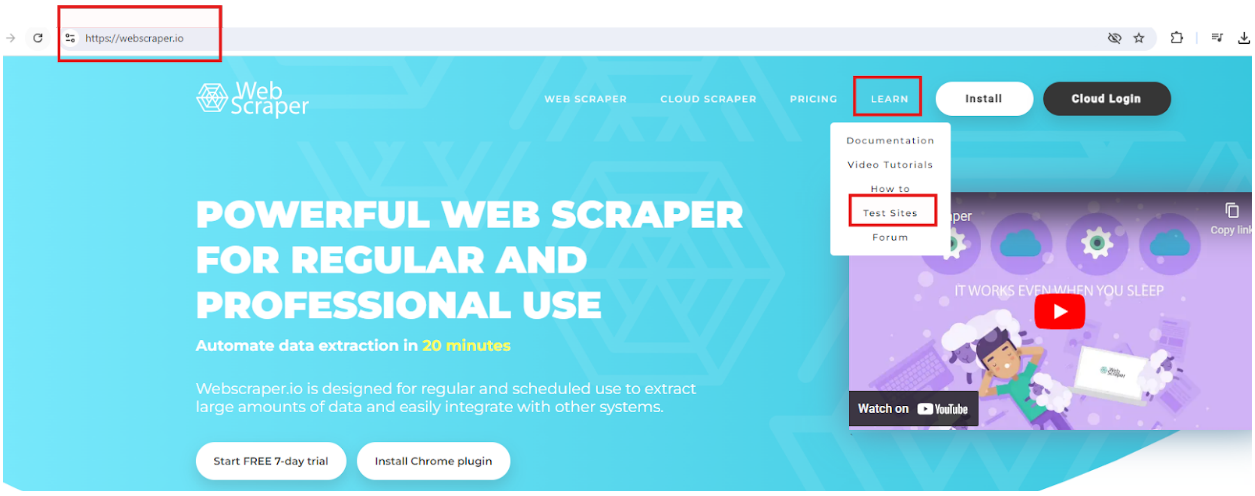
Project description

Requests

Requests is a simple, yet elegant, HTTP library.

```
>>> import requests
>>> r = requests.get('https://httpbin.org/basic-auth/user/pass', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"authenticated": true, ...'
>>> r.json()
{'authenticated': True, ...}
```

2. Use the website, WEBSCRAPERS.IO for the TEST SITES.

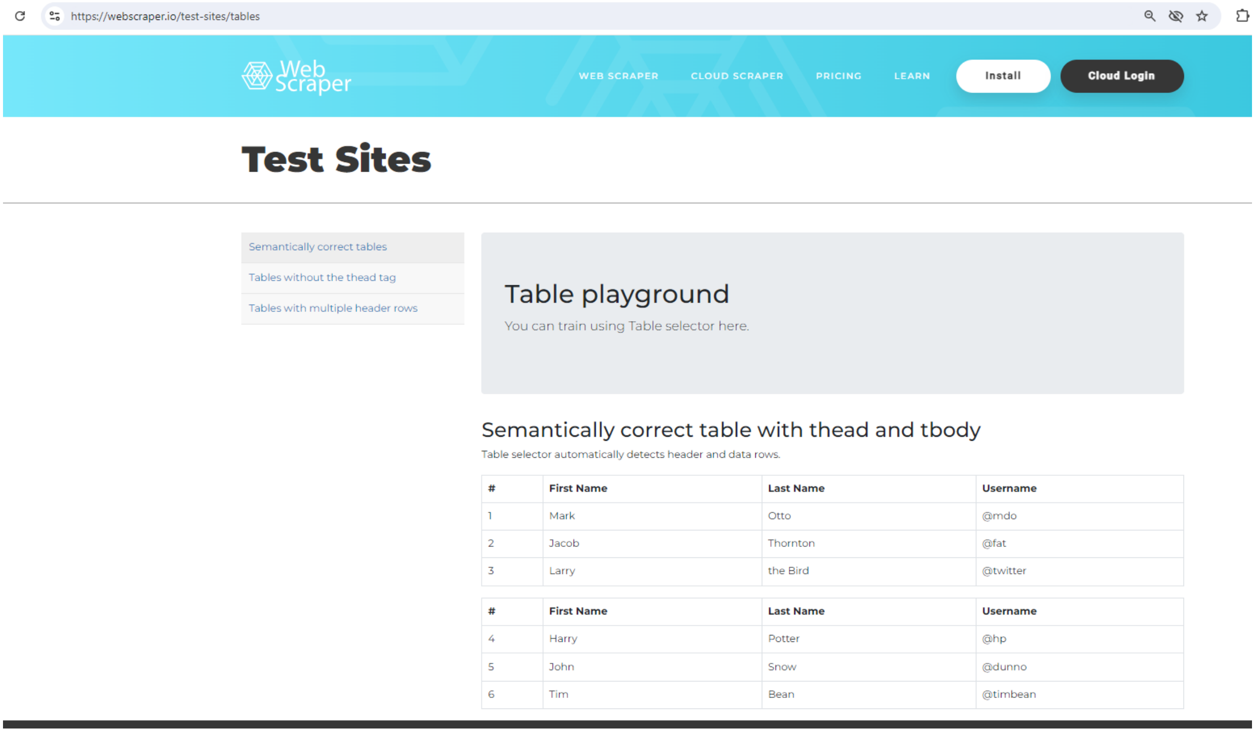


POWERFUL WEB SCRAPER FOR REGULAR AND PROFESSIONAL USE

Automate data extraction in 20 minutes

Webscraper.io is designed for regular and scheduled use to extract large amounts of data and easily integrate with other systems.

Use the TABLE PLAYGROUND for testing.


https://webscraper.io/test-sites/tables
Web Scraper
WEB SCRAPER
CLOUD SCRAPER
PRICING
LEARN
Install
Cloud Login
Semantically correct tables
Tables without the thead tag
Tables with multiple header rows

Test Sites

Table playground

You can train using Table selector here.

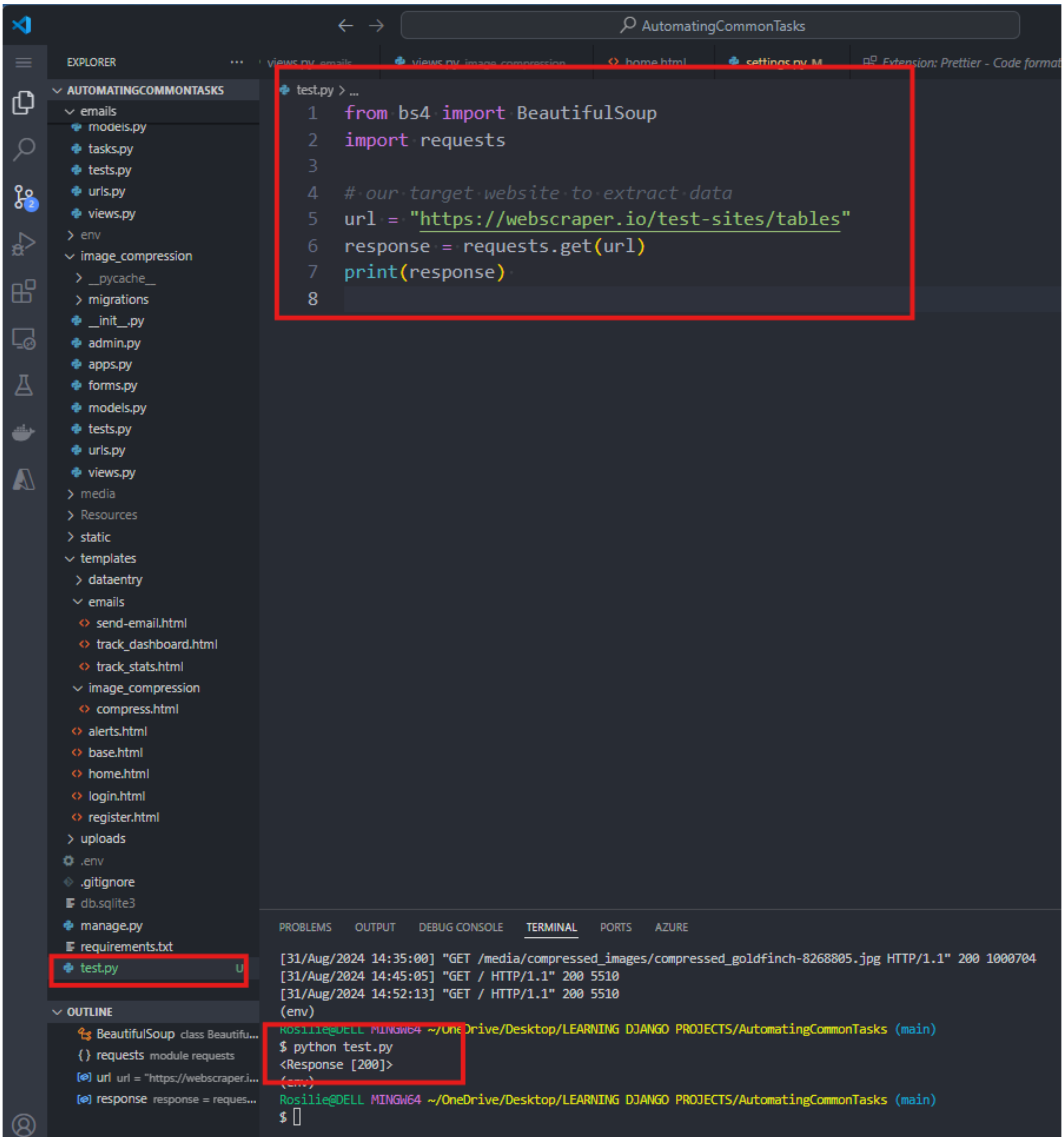
Semantically correct table with thead and tbody

Table selector automatically detects header and data rows.

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

#	First Name	Last Name	Username
4	Harry	Potter	@hp
5	John	Snow	@dunno
6	Tim	Bean	@timbean

3. In the root directory, create a new file, TEST.PY and update:

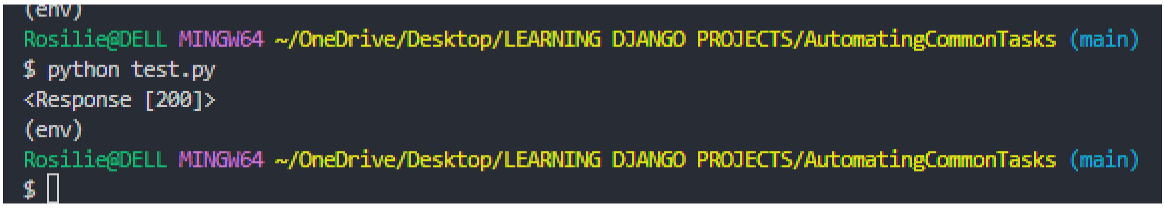


```python
from bs4 import BeautifulSoup
import requests

# our target website to extract data
url = "https://webscraper.io/test-sites/tables"
response = requests.get(url)
print(response)

```

```
[31/Aug/2024 14:35:00] "GET /media/compressed_images/compressed_goldfinch-8268805.jpg HTTP/1.1" 200 1000704
[31/Aug/2024 14:45:05] "GET / HTTP/1.1" 200 5510
[31/Aug/2024 14:52:13] "GET / HTTP/1.1" 200 5510
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<Response [200]>
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$
```

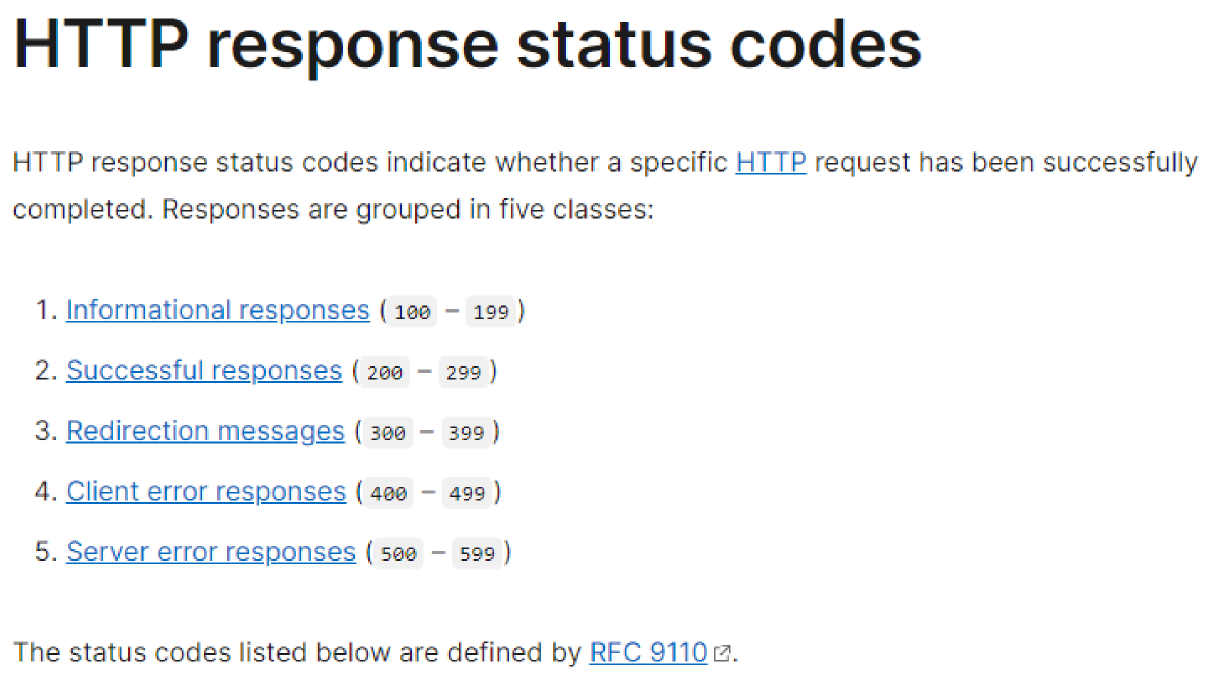
4. Run this file using and it returns 200 - MEANING OK.

$ python test.py



```
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<Response [200]>
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$
```

5. Use this as your guide for HTTP RESPONSES:



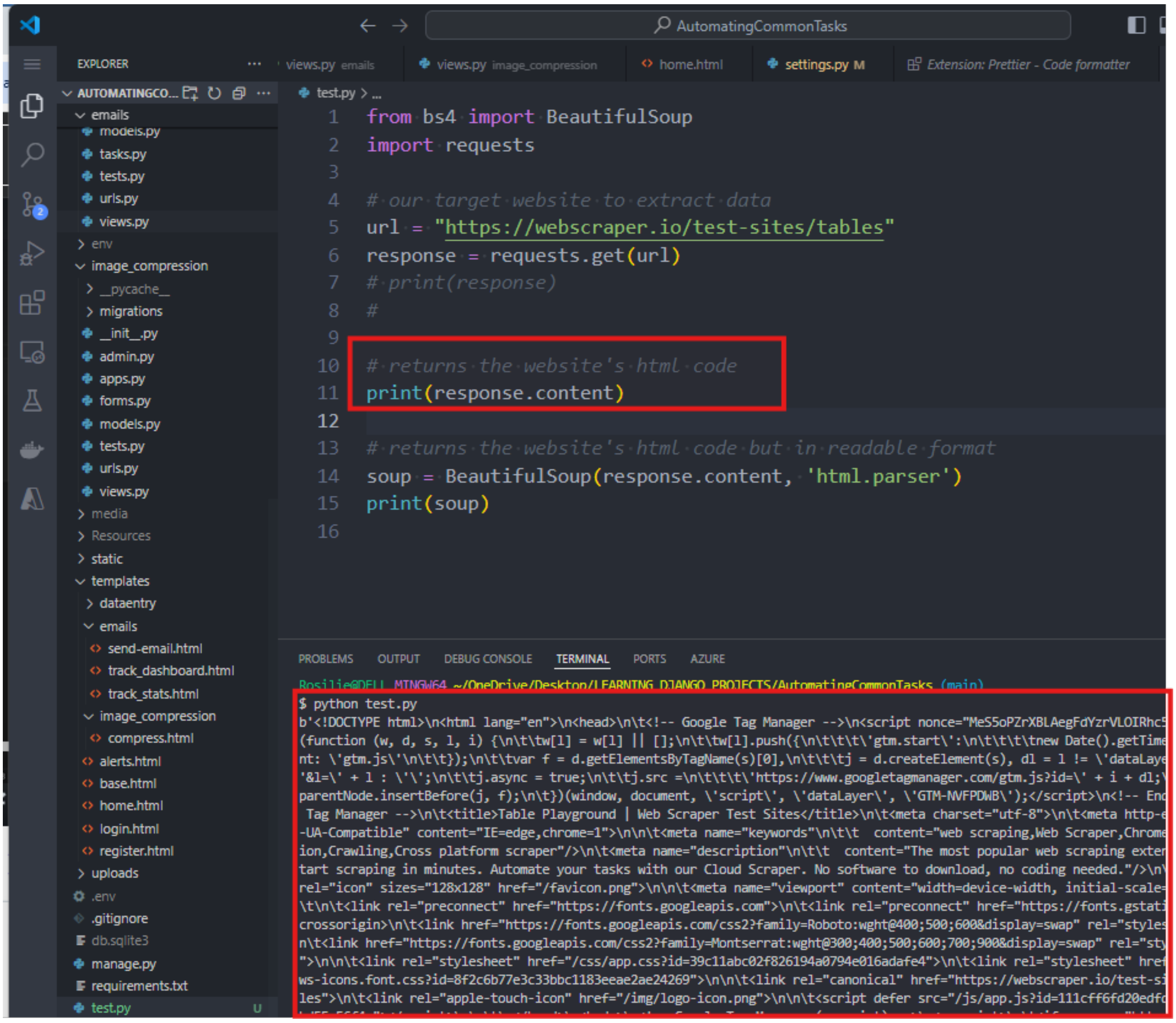
HTTP response status codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (`100` – `199`)
2. Successful responses (`200` – `299`)
3. Redirection messages (`300` – `399`)
4. Client error responses (`400` – `499`)
5. Server error responses (`500` – `599`)

The status codes listed below are defined by RFC 9110.

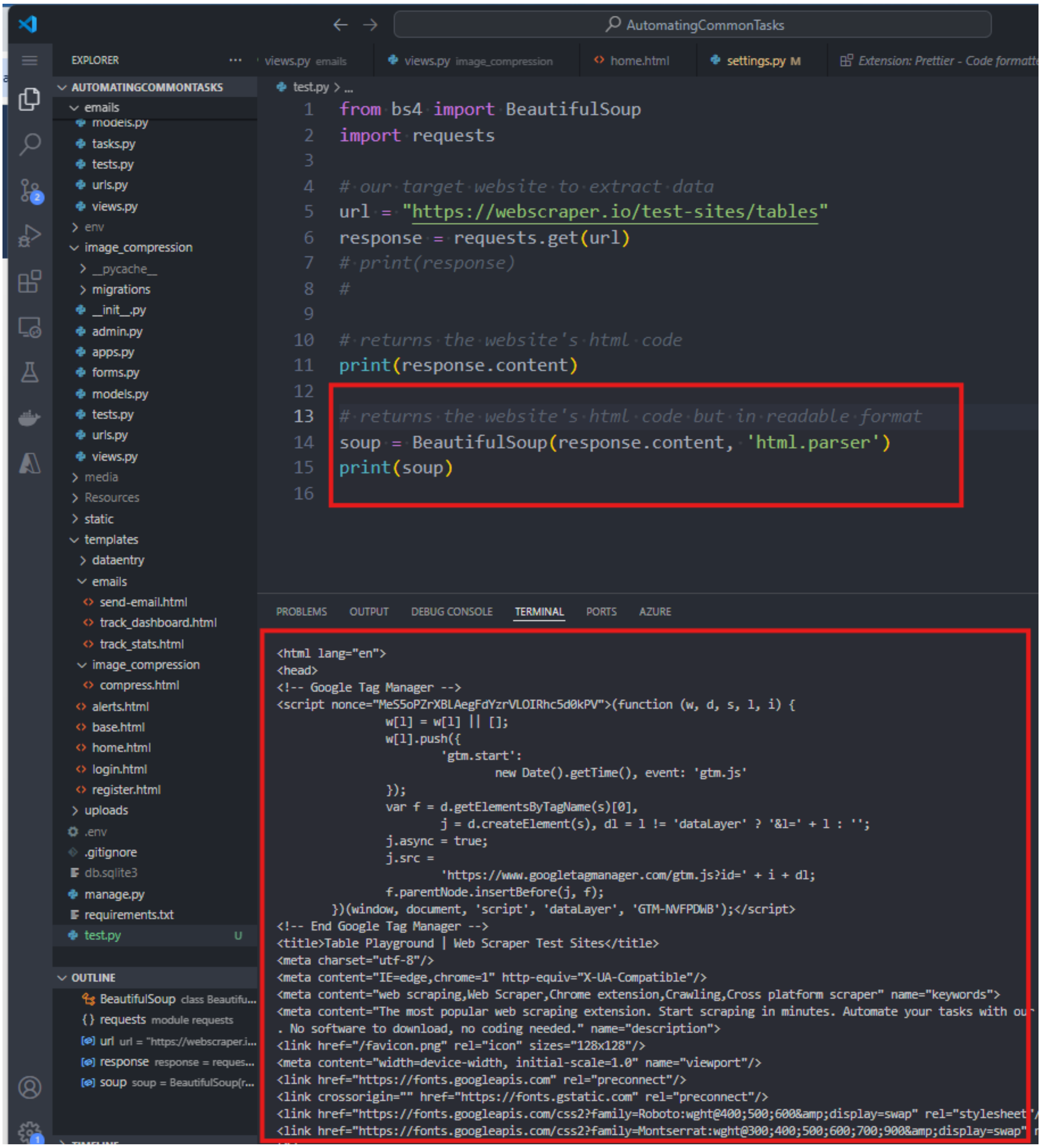
6. To get the website's HTML code, we type:



```python
from bs4 import BeautifulSoup
import requests

# our target website to extract data
url = "https://webscraper.io/test-sites/tables"
response = requests.get(url)
# print(response)
#

# returns the website's html code
print(response.content)

# returns the website's html code but in readable format
soup = BeautifulSoup(response.content, 'html.parser')
print(soup)

```

```
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
b'<!DOCTYPE html>\n<html lang="en">\n<head>\n\t<!-- Google Tag Manager -->\n<script nonce="MeS5oPZrXBLAegFdYzrVLOIRhc5
(function (w, d, s, l, i) {\n\t\tw[l] = w[l] || [];\n\t\tw[l].push({\n\t\t\t\'gtm.start\':\n\t\t\t\tnew Date().getTime
nt: \'gtm.js\'\n\t\t});\n\t\tvar f = d.getElementsByTagName(s)[0],\n\t\t\tj = d.createElement(s), dl = l != \'dataLaye
'&l=\' + l : \'\';\n\t\tj.async = true;\n\t\tj.src =\n\t\t\t\'https://www.googletagmanager.com/gtm.js?id=\' + i + dl;\
parentNode.insertBefore(j, f);\n\t})(window, document, \'script\', \'dataLayer\', \'GTM-NVFPDWB\');</script>\n<!-- End
 Tag Manager -->\n\t<title>Table Playground | Web Scraper Test Sites</title>\n\t<meta charset="utf-8">\n\t<meta http-e
-UA-Compatible" content="IE=edge,chrome=1">\n\n\t<meta name="keywords"\n\t\t  content="web scraping,Web Scraper,Chrome
ion,Crawling,Cross platform scraper"/>\n\t<meta name="description"\n\t\t  content="The most popular web scraping exter
tart scraping in minutes. Automate your tasks with our Cloud Scraper. No software to download, no coding needed."/>\n\
rel="icon" sizes="128x128" href="/favicon.png">\n\n\t<meta name="viewport" content="width=device-width, initial-scale=
\t\n\t<link rel="preconnect" href="https://fonts.googleapis.com">\n\t<link rel="preconnect" href="https://fonts.gstati
crossorigin>\n\t<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;600&display=swap" rel="styles
n\t<link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;500;600;700;900&display=swap" rel="sty
">\n\n\t<link rel="stylesheet" href="/css/app.css?id=39c11abc02f826194a0794e016adafe4">\n\t<link rel="stylesheet" href
ws-icons.font.css?id=8f2c6b77e3c33bbc1183eeae2ae24269">\n\n\t<link rel="canonical" href="https://webscraper.io/test-si
les">\n\t<link rel="apple-touch-icon" href="/img/logo-icon.png">\n\n\t<script defer src="/js/app.js?id=111cff6fd20edfc
```

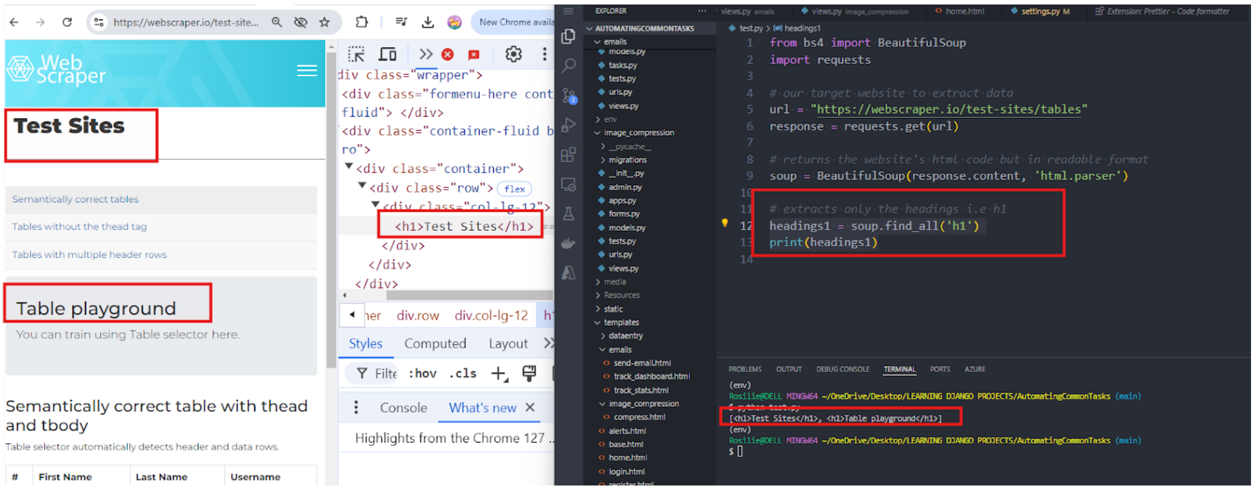
7. If we want to read the website's code in a more readable format, we can use BeautifulSoup.



```python
from bs4 import BeautifulSoup
import requests

# our target website to extract data
url = "https://webscraper.io/test-sites/tables"
response = requests.get(url)
# print(response)
#

# returns the website's html code
print(response.content)

# returns the website's html code but in readable format
soup = BeautifulSoup(response.content, 'html.parser')
print(soup)

```

```
<html lang="en">
<head>
<!-- Google Tag Manager -->
<script nonce="MeS5oPZrXBLAegFdYzrVLOIRhc5d0kPV">(function (w, d, s, l, i) {
                w[l] = w[l] || [];
                w[l].push({
                        'gtm.start':
                                new Date().getTime(), event: 'gtm.js'
                });
                var f = d.getElementsByTagName(s)[0],
                        j = d.createElement(s), dl = l != 'dataLayer' ? '&l=' + l : '';
                j.async = true;
                j.src =
                        'https://www.googletagmanager.com/gtm.js?id=' + i + dl;
                f.parentNode.insertBefore(j, f);
        })(window, document, 'script', 'dataLayer', 'GTM-NVFPDWB');</script>
<!-- End Google Tag Manager -->
<title>Table Playground | Web Scraper Test Sites</title>
<meta charset="utf-8"/>
<meta content="IE=edge,chrome=1" http-equiv="X-UA-Compatible"/>
<meta content="web scraping,Web Scraper,Chrome extension,Crawling,Cross platform scraper" name="keywords">
<meta content="The most popular web scraping extension. Start scraping in minutes. Automate your tasks with ou
. No software to download, no coding needed." name="description">
<link href="/favicon.png" rel="icon" sizes="128x128"/>
<meta content="width=device-width, initial-scale=1.0" name="viewport"/>
<link href="https://fonts.googleapis.com" rel="preconnect"/>
<link crossorigin="" href="https://fonts.gstatic.com" rel="preconnect"/>
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;600&amp;display=swap" rel="stylesheet"/
<link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;500;600;700;900&amp;display=swap" r
```

8. To extract only certain portions of the website like H1 headings only



Test Sites

Semantically correct tables

Tables without the thead tag

Tables with multiple header rows

Table playground

You can train using Table selector here.

Semantically correct table with thead and tbody

Table selector automatically detects header and data rows.

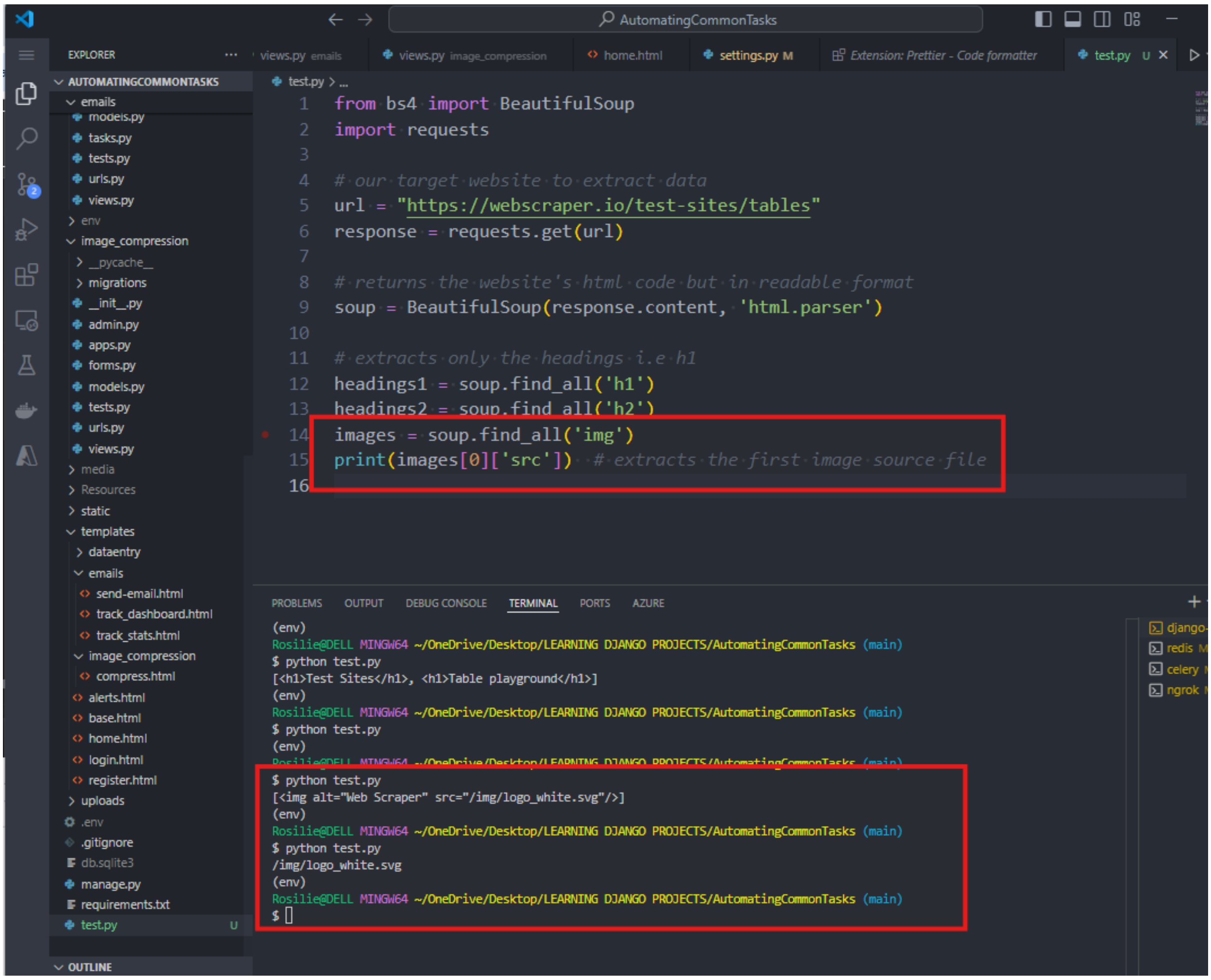
#	First Name	Last Name	Username

```html
div class="wrapper">
 <div class="formenu-here cont
 fluid"> </div>
 <div class="container-fluid b
 ro">
  <div class="container">
   <div class="row"> flex
    <div class="col-lg-12">
      <h1>Test Sites</h1>
    </div>
   </div>
  </div>
```

```python
from bs4 import BeautifulSoup
import requests

# our target website to extract data
url = "https://webscraper.io/test-sites/tables"
response = requests.get(url)

# returns the website's html code but in readable format
soup = BeautifulSoup(response.content, 'html.parser')

# extracts only the headings i.e h1
headings1 = soup.find_all('h1')
print(headings1)

```

```
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
[<h1>Test Sites</h1>, <h1>Table playground</h1>]
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ 
```

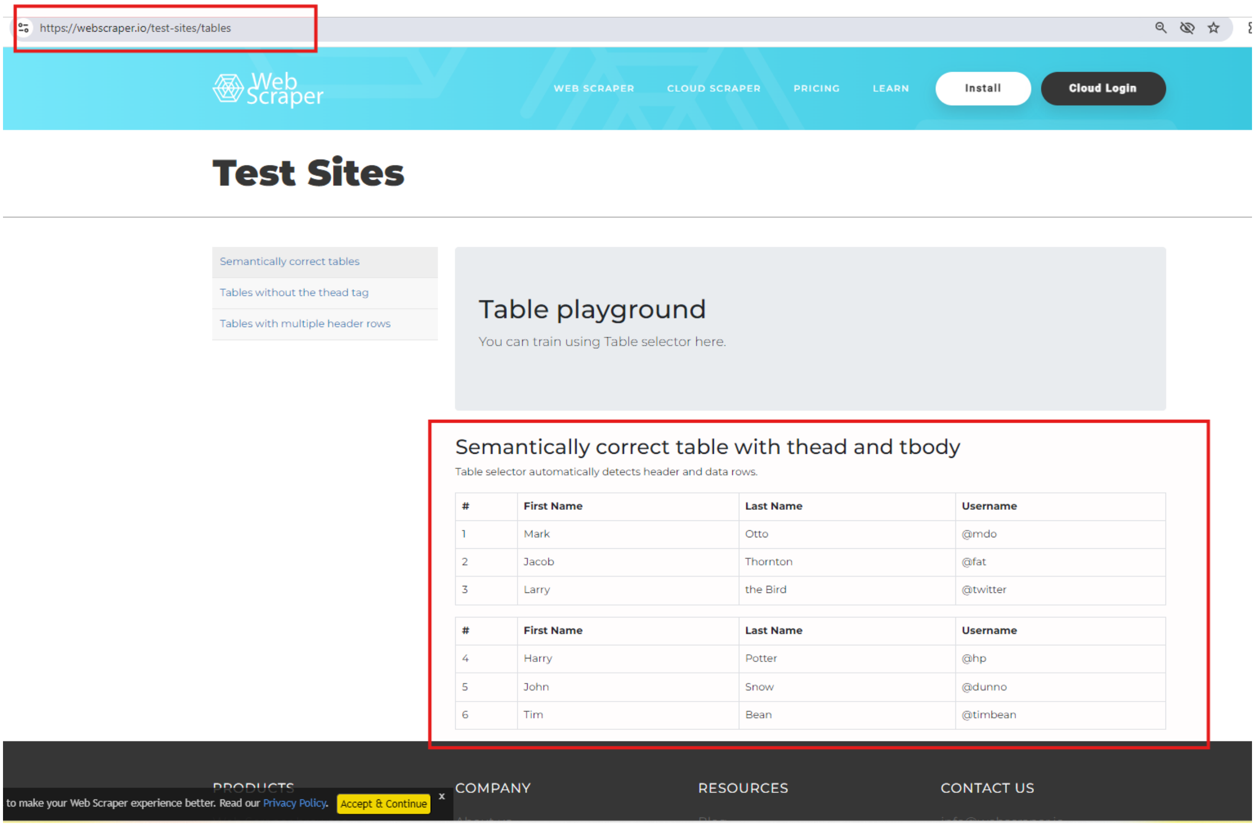
9. To extract the images:



```python
from bs4 import BeautifulSoup
import requests

# our target website to extract data
url = "https://webscraper.io/test-sites/tables"
response = requests.get(url)

# returns the website's html code but in readable format
soup = BeautifulSoup(response.content, 'html.parser')

# extracts only the headings i.e h1
headings1 = soup.find_all('h1')
headings2 = soup.find_all('h2')
images = soup.find_all('img')
print(images[0]['src'])  # extracts the first image source file
```

```
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
[<h1>Test Sites</h1>, <h1>Table playground</h1>]
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
[<img alt="Web Scraper" src="/img/logo_white.svg"/>]
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
/img/logo_white.svg
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ 
```

10. To extract the information from the website's tables:



Test Sites

Semantically correct tables
Tables without the thead tag
Tables with multiple header rows

Table playground

You can train using Table selector here.

Semantically correct table with thead and tbody

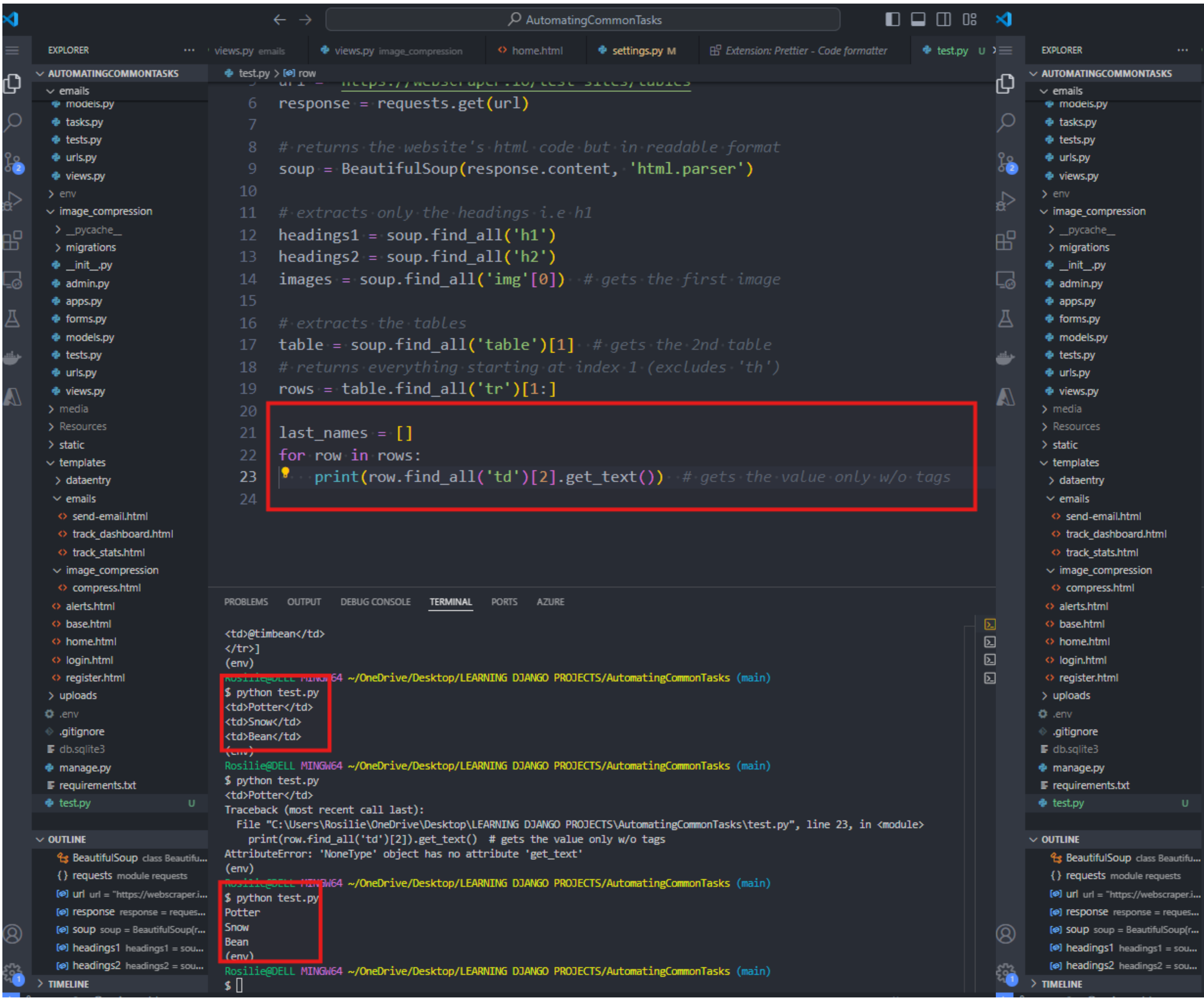
Table selector automatically detects header and data rows.

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

#	First Name	Last Name	Username
4	Harry	Potter	@hp
5	John	Snow	@dunno
6	Tim	Bean	@timbean

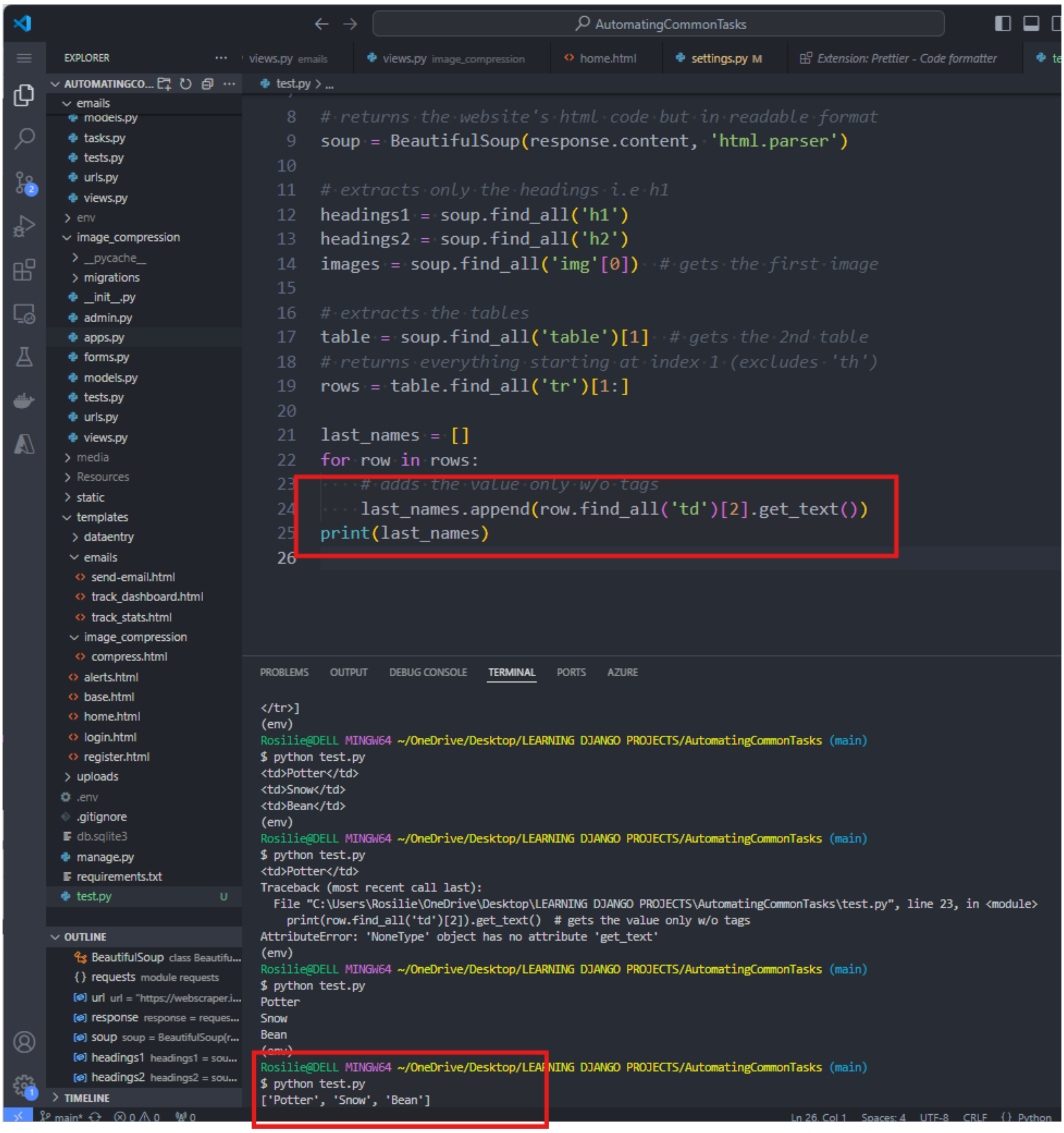
PRODUCTS
COMPANY
RESOURCES
CONTACT US
to make your Web Scraper experience better. Read our Privacy Policy.
Accept & Continue

11. Extract only the LASTNAME OF THE SECOND TABLE:



```python
response = requests.get(url)

# returns the website's html code but in readable format
soup = BeautifulSoup(response.content, 'html.parser')

# extracts only the headings i.e h1
headings1 = soup.find_all('h1')
headings2 = soup.find_all('h2')
images = soup.find_all('img'[0])  # gets the first image

# extracts the tables
table = soup.find_all('table')[1]  # gets the 2nd table
# returns everything starting at index 1 (excludes 'th')
rows = table.find_all('tr')[1:]

last_names = []
for row in rows:
    print(row.find_all('td')[2].get_text())  # gets the value only w/o tags
```

```
<td>@timbean</td>
</tr>]
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<td>Potter</td>
<td>Snow</td>
<td>Bean</td>
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<td>Potter</td>
Traceback (most recent call last):
  File "C:\Users\Rosilie\OneDrive\Desktop\LEARNING DJANGO PROJECTS\AutomatingCommonTasks\test.py", line 23, in <module>
    print(row.find_all('td')[2]).get_text()  # gets the value only w/o tags
AttributeError: 'NoneType' object has no attribute 'get_text'
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
Potter
Snow
Bean
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ 
```

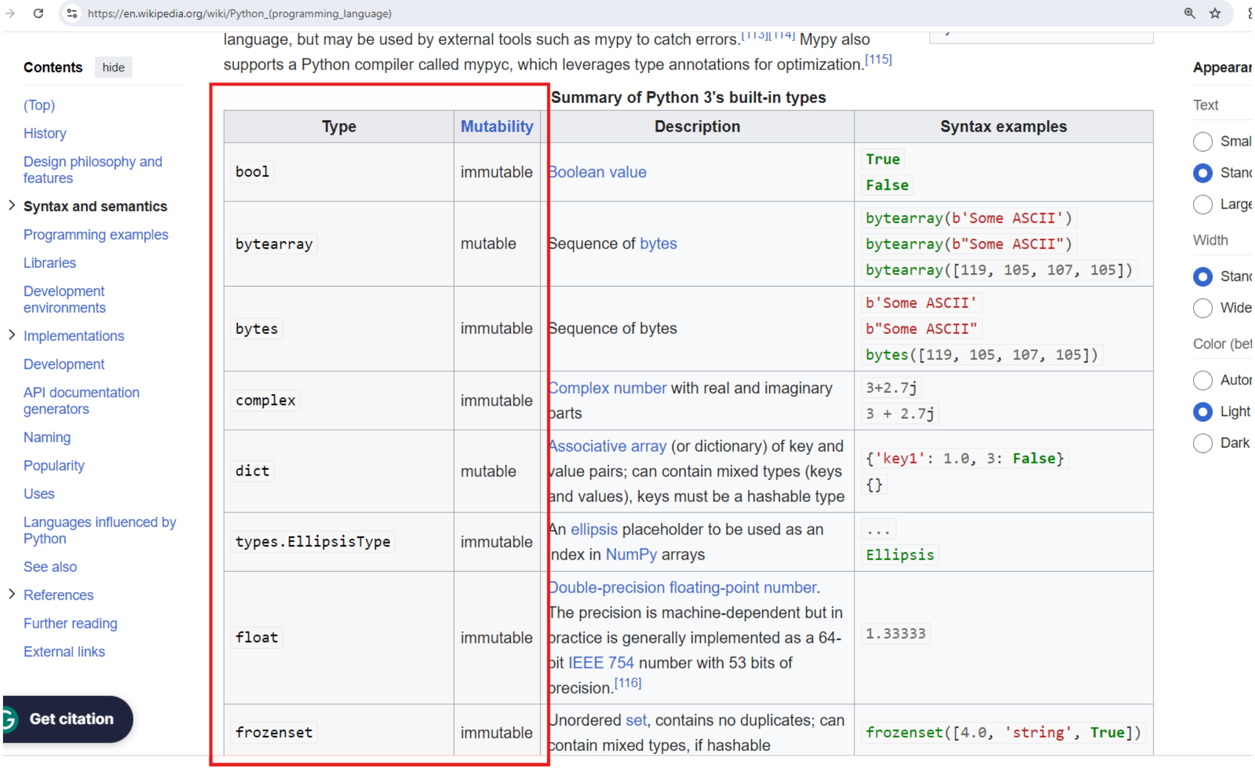


```python
# returns the website's html code but in readable format
soup = BeautifulSoup(response.content, 'html.parser')

# extracts only the headings i.e h1
headings1 = soup.find_all('h1')
headings2 = soup.find_all('h2')
images = soup.find_all('img'[0])  # gets the first image

# extracts the tables
table = soup.find_all('table')[1]  # gets the 2nd table
# returns everything starting at index 1 (excludes 'th')
rows = table.find_all('tr')[1:]

last_names = []
for row in rows:
    # adds the value only w/o tags
    last_names.append(row.find_all('td')[2].get_text())
print(last_names)

```

```
</tr>]
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<td>Potter</td>
<td>Snow</td>
<td>Bean</td>
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<td>Potter</td>
Traceback (most recent call last):
  File "C:\Users\Rosilie\OneDrive\Desktop\LEARNING DJANGO PROJECTS\AutomatingCommonTasks\test.py", line 23, in <module>
    print(row.find_all('td')[2]).get_text()  # gets the value only w/o tags
AttributeError: 'NoneType' object has no attribute 'get_text'
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
Potter
Snow
Bean
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
['Potter', 'Snow', 'Bean']
```

WEBSITE # 2: WIKIPEDIA ON PYTHON

The goal is to categorize the data types according to MUTABLE OR IMMUTABLE



https://en.wikipedia.org/wiki/Python_(programming_language)

Contents hide

(Top)
History
Design philosophy and features
Syntax and semantics
Programming examples
Libraries
Development environments
Implementations
Development
API documentation generators
Naming
Popularity
Uses
Languages influenced by Python
See also
References
Further reading
External links

Get citation

language, but may be used by external tools such as mypy to catch errors.[113][114] Mypy also supports a Python compiler called mypyc, which leverages type annotations for optimization.[115]

Summary of Python 3's built-in types

Type	Mutability	Description	Syntax examples
`bool`	immutable	Boolean value	`True` `False`

`bytearray`	mutable	Sequence of bytes	`bytearray(b'Some ASCII')`
			`bytearray(b"Some ASCII")`
			`bytearray([119, 105, 107, 105])`

`bytes`	immutable	Sequence of bytes	`b'Some ASCII'`
			`b"Some ASCII"`
			`bytes([119, 105, 107, 105])`

`complex`	immutable	Complex number with real and imaginary parts	`3+2.7j`
			`3 + 2.7j`

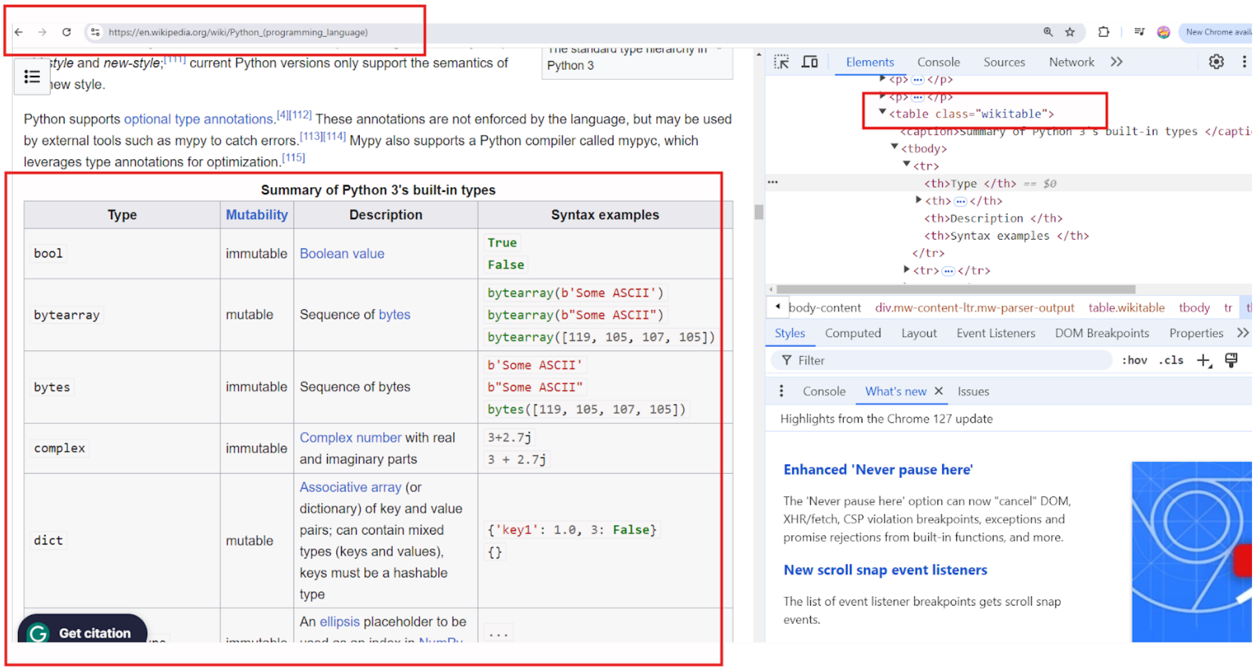
`dict`	mutable	Associative array (or dictionary) of key and value pairs; can contain mixed types (keys and values), keys must be a hashable type	`{'key1': 1.0, 3: False}`
			`{}`

`types.EllipsisType`	immutable	An ellipsis placeholder to be used as an index in NumPy arrays	`...`
			`Ellipsis`

`float`	immutable	Double-precision floating-point number. The precision is machine-dependent but in practice is generally implemented as a 64-bit IEEE 754 number with 53 bits of precision.[116]	`1.33333`
`frozenset`	immutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	`frozenset([4.0, 'string', True])`

Appearance
Text
○ Small
● Standard
○ Large
Width
● Standard
○ Wide
Color (beta)
○ Automatic
● Light
○ Dark

12. We can also use the CLASS name to find an element on the website. So, the table uses a classname 'WIKIPEDIA'



https://en.wikipedia.org/wiki/Python_(programming_language)

style and *new-style*;[111] current Python versions only support the semantics of new style.

The standard type hierarchy in Python 3

Python supports optional type annotations.[4][112] These annotations are not enforced by the language, but may be used by external tools such as mypy to catch errors.[113][114] Mypy also supports a Python compiler called mypyc, which leverages type annotations for optimization.[115]

Summary of Python 3's built-in types

Type	Mutability	Description	Syntax examples
`bool`	immutable	Boolean value	`True` `False`

`bytearray`	mutable	Sequence of bytes	`bytearray(b'Some ASCII')`
			`bytearray(b"Some ASCII")`
			`bytearray([119, 105, 107, 105])`

`bytes`	immutable	Sequence of bytes	`b'Some ASCII'`
			`b"Some ASCII"`
			`bytes([119, 105, 107, 105])`

`complex`	immutable	Complex number with real and imaginary parts	`3+2.7j`
			`3 + 2.7j`

`dict`	mutable	Associative array (or dictionary) of key and value pairs; can contain mixed types (keys and values), keys must be a hashable type	`{'key1': 1.0, 3: False}`
			`{}`

	immutable	An ellipsis placeholder to be used as an index in NumPy	`...`

Get citation

Elements Console Sources Network

```
▶<p> … </p>
▶<p> … </p>
▼<table class="wikitable">
   <caption>Summary of Python 3's built-in types </capti
  ▼<tbody>
    ▼<tr>
       <th>Type </th> == $0
      ▶<th> … </th>
       <th>Description </th>
       <th>Syntax examples </th>
     </tr>
    ▶<tr> … </tr>
```

body-content div.mw-content-ltr.mw-parser-output table.wikitable tbody tr th

Styles Computed Layout Event Listeners DOM Breakpoints Properties

Filter :hov .cls

Console What's new Issues

Highlights from the Chrome 127 update

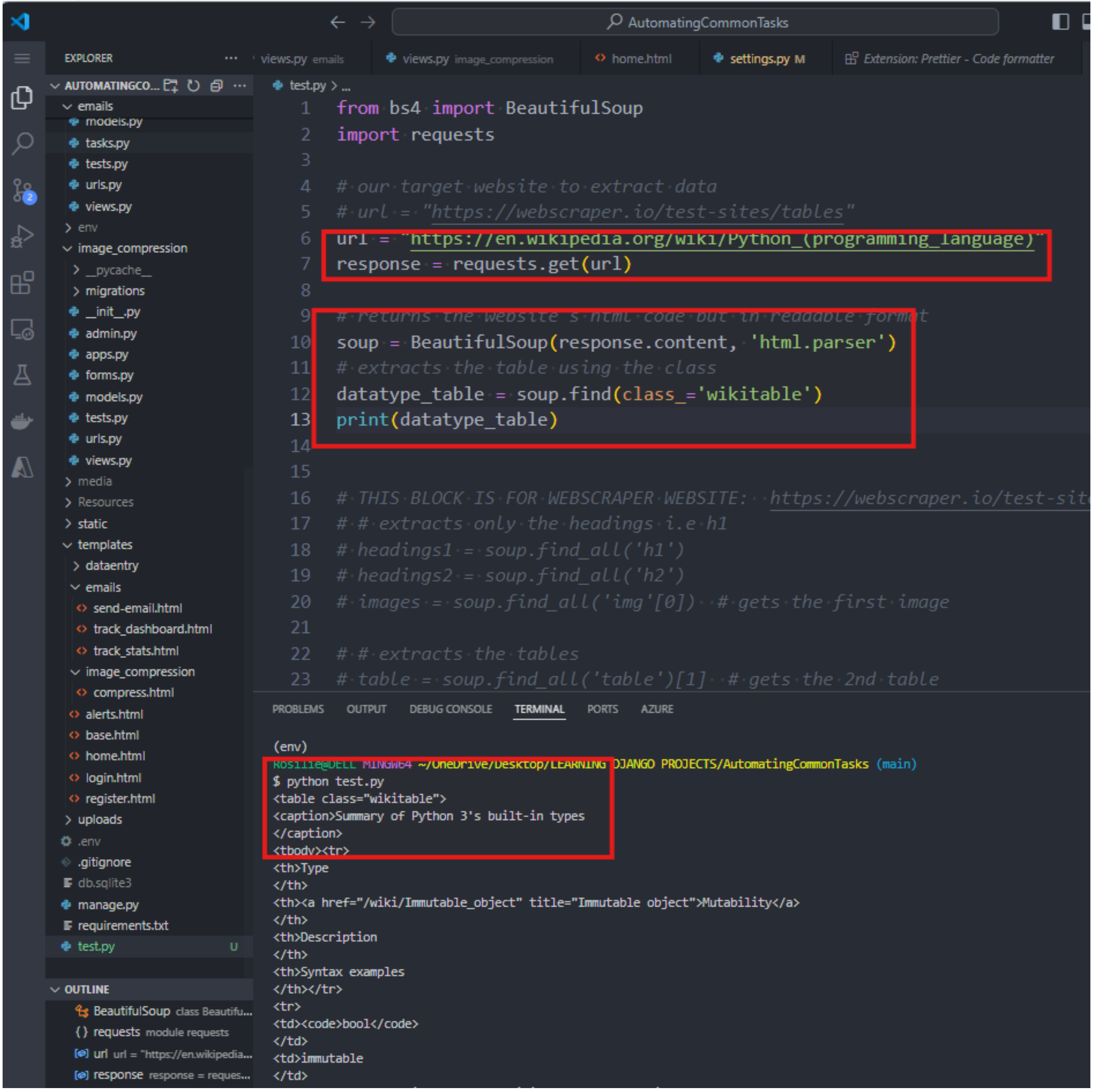
Enhanced 'Never pause here'

The 'Never pause here' option can now "cancel" DOM, XHR/fetch, CSP violation breakpoints, exceptions and promise rejections from built-in functions, and more.

New scroll snap event listeners

The list of event listener breakpoints gets scroll snap events.

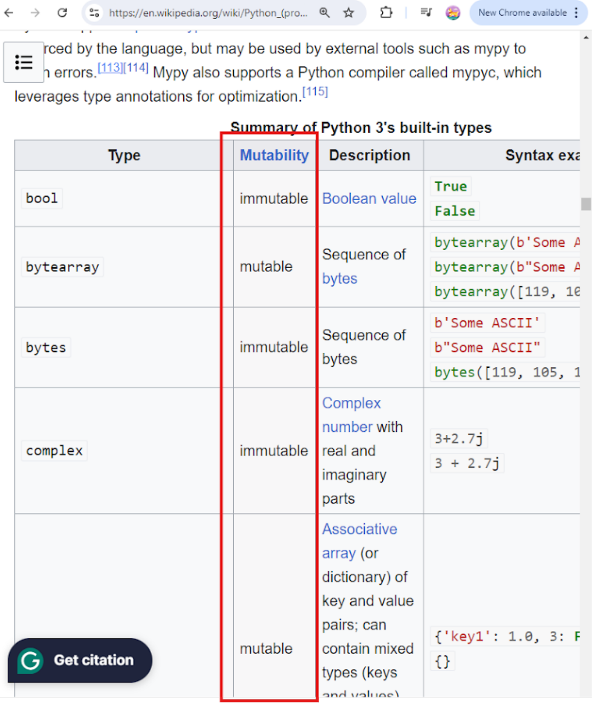
We can update our TEST.PY AS:



```python
from bs4 import BeautifulSoup
import requests

# our target website to extract data
# url = "https://webscraper.io/test-sites/tables"
url = "https://en.wikipedia.org/wiki/Python_(programming_language)"
response = requests.get(url)

# returns the website's html code but in readable format
soup = BeautifulSoup(response.content, 'html.parser')
# extracts the table using the class
datatype_table = soup.find(class_='wikitable')
print(datatype_table)


# THIS BLOCK IS FOR WEBSCRAPER WEBSITE:  https://webscraper.io/test-sit
# # extracts only the headings i.e h1
# headings1 = soup.find_all('h1')
# headings2 = soup.find_all('h2')
# images = soup.find_all('img'[0])  # gets the first image

# # extracts the tables
# table = soup.find_all('table')[1]  # gets the 2nd table
```

```
(env)
Rosiie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<table class="wikitable">
<caption>Summary of Python 3's built-in types
</caption>
<tbody><tr>
<th>Type
</th>
<th><a href="/wiki/Immutable_object" title="Immutable object">Mutability</a>
</th>
<th>Description
</th>
<th>Syntax examples
</th></tr>
<tr>
<td><code>bool</code>
</td>
<td>immutable
</td>
```

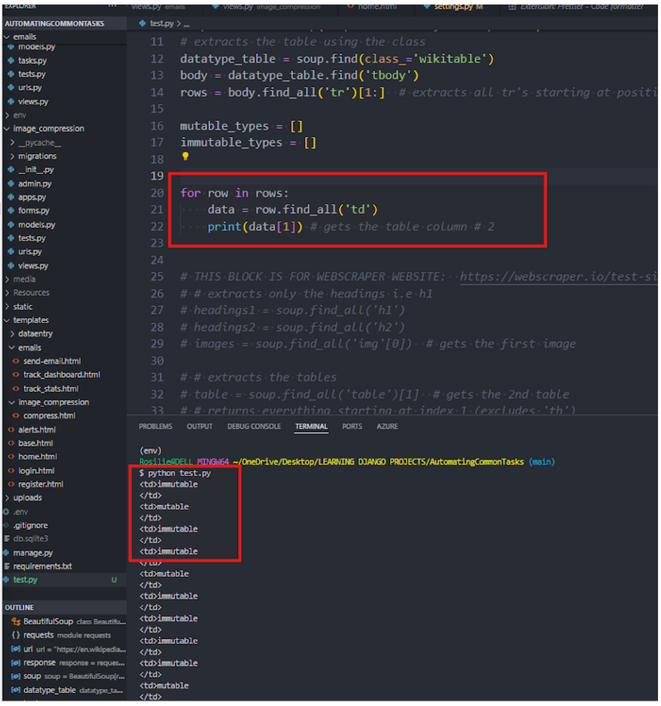
13. Each data type has a tag 'TD' inside the 'TBODY.' We need to use this info then in our TEST.PY. Each table column can be accessed using an INDEX POSITION where INDEX 0 means our first column, INDEX 1 means our second column.



rced by the language, but may be used by external tools such as mypy to n errors.[113][114] Mypy also supports a Python compiler called mypyc, which leverages type annotations for optimization.[115]

Summary of Python 3's built-in types

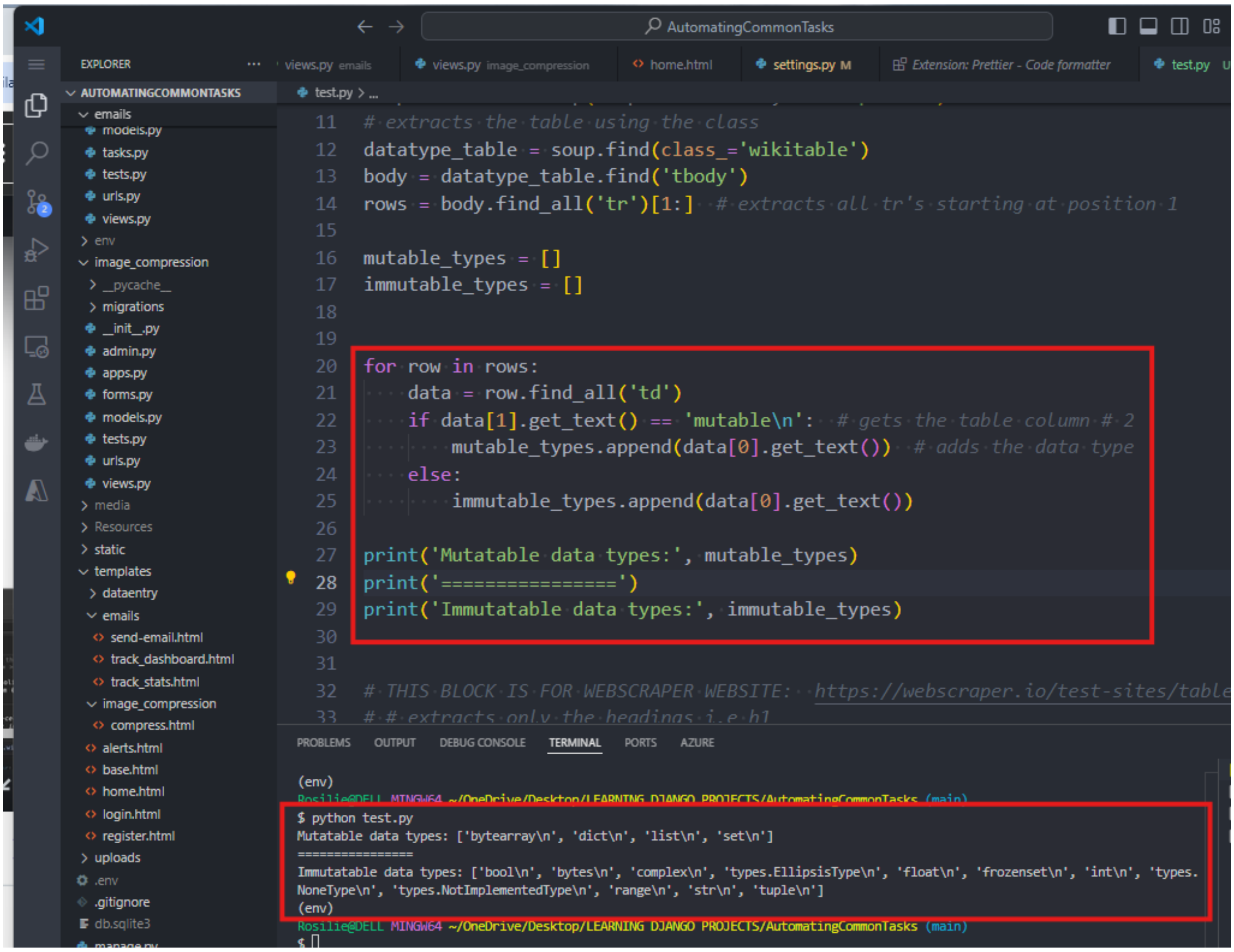
Type	Mutability	Description	Syntax exa
bool	immutable	Boolean value	True False
bytearray	mutable	Sequence of bytes	bytearray(b'Some A bytearray(b"Some A bytearray([119, 10
bytes	immutable	Sequence of bytes	b'Some ASCII' b"Some ASCII" bytes([119, 105, 1
complex	immutable	Complex number with real and imaginary parts	3+2.7j 3 + 2.7j
	mutable	Associative array (or dictionary) of key and value pairs; can contain mixed types (keys and values)	{'key1': 1.0, 3: F {}



```python
# extracts the table using the class
datatype_table = soup.find(class_='wikitable')
body = datatype_table.find('tbody')
rows = body.find_all('tr')[1:]  # extracts all tr's starting at positi

mutable_types = []
immutable_types = []


for row in rows:
    data = row.find_all('td')
    print(data[1]) # gets the table column # 2


# THIS BLOCK IS FOR WEBSCRAPER WEBSITE:  https://webscraper.io/test-si
# # extracts only the headings i.e h1
# headings1 = soup.find_all('h1')
# headings2 = soup.find_all('h2')
# images = soup.find_all('img'[0])  # gets the first image

# # extracts the tables
# table = soup.find_all('table')[1]  # gets the 2nd table
# # returns everything starting at index 1 (excludes 'th')
```

```
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<td>immutable
</td>
<td>mutable
</td>
<td>immutable
</td>
<td>immutable
</td>
<td>mutable
</td>
<td>immutable
</td>
<td>immutable
</td>
<td>immutable
</td>
<td>immutable
</td>
<td>mutable
</td>
```

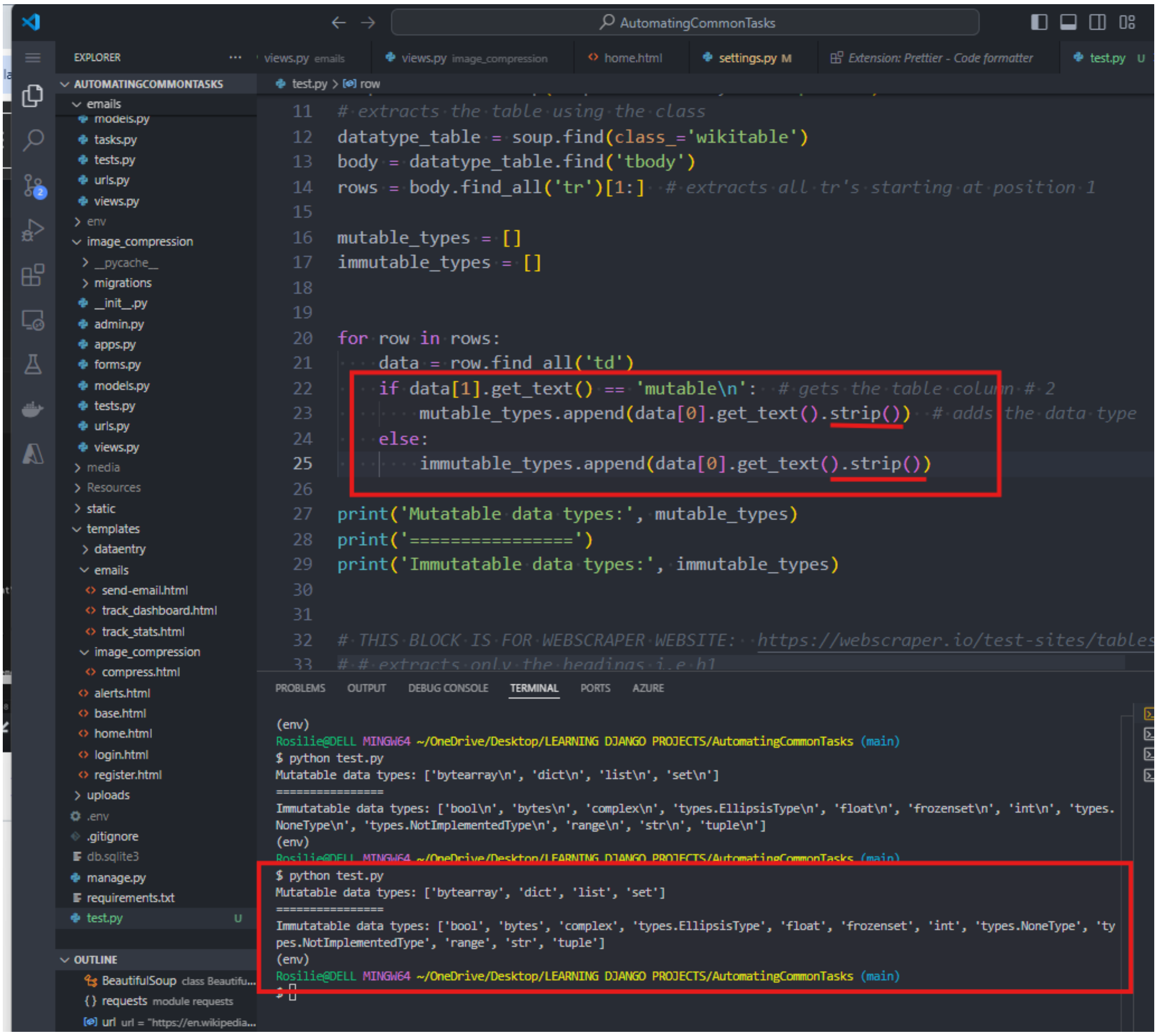
14. Update TEST.PY as:



```python
# extracts the table using the class
datatype_table = soup.find(class_='wikitable')
body = datatype_table.find('tbody')
rows = body.find_all('tr')[1:]  # extracts all tr's starting at position 1

mutable_types = []
immutable_types = []


for row in rows:
    data = row.find_all('td')
    if data[1].get_text() == 'mutable\n':  # gets the table column # 2
        mutable_types.append(data[0].get_text())  # adds the data type
    else:
        immutable_types.append(data[0].get_text())

print('Mutatable data types:', mutable_types)
print('===============')
print('Immutatable data types:', immutable_types)


# THIS BLOCK IS FOR WEBSCRAPER WEBSITE:  https://webscraper.io/test-sites/table
# # extracts only the headings i.e h1
```

```
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
Mutatable data types: ['bytearray\n', 'dict\n', 'list\n', 'set\n']
===============
Immutatable data types: ['bool\n', 'bytes\n', 'complex\n', 'types.EllipsisType\n', 'float\n', 'frozenset\n', 'int\n', 'types.
NoneType\n', 'types.NotImplementedType\n', 'range\n', 'str\n', 'tuple\n']
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ 
```

15. To remove the NEWLINE (\n), we can use the STRIP FUNCTION:



```python
# extracts the table using the class
datatype_table = soup.find(class_='wikitable')
body = datatype_table.find('tbody')
rows = body.find_all('tr')[1:]  # extracts all tr's starting at position 1

mutable_types = []
immutable_types = []


for row in rows:
    data = row.find_all('td')
    if data[1].get_text() == 'mutable\n':  # gets the table column # 2
        mutable_types.append(data[0].get_text().strip())  # adds the data type
    else:
        immutable_types.append(data[0].get_text().strip())

print('Mutatable data types:', mutable_types)
print('================')
print('Immutatable data types:', immutable_types)


# THIS BLOCK IS FOR WEBSCRAPER WEBSITE:  https://webscraper.io/test-sites/tables
# # extracts only the headings i.e h1
```

```
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
Mutatable data types: ['bytearray\n', 'dict\n', 'list\n', 'set\n']
================
Immutatable data types: ['bool\n', 'bytes\n', 'complex\n', 'types.EllipsisType\n', 'float\n', 'frozenset\n', 'int\n', 'types.
NoneType\n', 'types.NotImplementedType\n', 'range\n', 'str\n', 'tuple\n']
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
Mutatable data types: ['bytearray', 'dict', 'list', 'set']
================
Immutatable data types: ['bool', 'bytes', 'complex', 'types.EllipsisType', 'float', 'frozenset', 'int', 'types.NoneType', 'ty
pes.NotImplementedType', 'range', 'str', 'tuple']
(env)
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$
```

16. These information are needed for STOCK MARKET ANALYSIS.