

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
```

AutomatingCommonTasks

EXPLORER

- AUTOMATINGCOMMONTASKS
 - emails
 - forms.py
 - models.py
 - tasks.py
 - tests.py
 - urls.py
 - views.py
 - env
 - image_compression
 - __pycache__
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - forms.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - media
 - Resources
 - static
 - templates
 - dataentry
 - emails
 - send-email.html
 - track_dashboard.html
 - track_stats.html
 - image_compression
 - compress.html
 - alerts.html
 - base.html
 - home.html
 - login.html
 - register.html
 - uploads
 - .env
 - .gitignore
 - db.sqlite3
 - manage.py
 - requirements.txt
- OUTLINE

No symbols found in document 'requirements.txt'

requirements.txt

```
1 amqp==5.2.0
2 asgiref==3.8.1
3 async-timeout==4.0.3
4 beautifulsoup4==4.12.3
5 billiard==4.2.0
6 celery==5.4.0
7 certifi==2024.7.4
8 charset-normalizer==3.3.2
9 click==8.1.7
10 click-didyoumean==0.3.1
11 click-plugins==1.1.1
12 click-repl==0.3.0
13 colorama==0.4.6
14 crispy-bootstrap5==2024.2
15 Django==4.2.14
16 django-anymail==11.1
17 django-ckeditor==6.7.1
18 django-crispy-forms==2.3
19 django-js-asset==2.2.0
20 idna==3.7
21 kombu==5.4.0
22 pillow==10.4.0
23 prompt_toolkit==3.0.47
24 python-dateutil==2.9.0.post0
25 python-decouple==3.8
26 redis==5.0.8
27 requests==2.32.3
28 six==1.16.0
29 soupsieve==2.6
30 sqlparse==0.5.1
31 typing_extensions==4.12.2
32 tzdata==2024.1
33 urllib3==2.2.2
34 vine==5.1.0
35 wcwidth==0.2.13
36
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

https://pypi.org/project/beautifulsoup4/

Search projects

Help Sponsors Log in Register

beautifulsoup4 4.12.3

pip install beautifulsoup4

Released: Jan 17, 2024

Screen-scraping library

Navigation

- Project description
- Release history

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.

https://pypi.org/project/requests/

Search projects

Help Sponsors Log in Register

requests 2.32.3

pip install requests

Released: May 29, 2024

Python HTTP for Humans.

Navigation

- Project description
- Release history
- Download files

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, ...}
```

Verified details [\(What is this?\)](#)
These details have been verified by PyPI

Maintainers

- graffatcolmingov
- Lukasa

2. Use the website, [WEBSCRAPERS.IO](https://webscrapers.io) for the TEST SITES.

https://webscraper.io

Web Scraper

WEB SCRAPER CLOUD SCRAPER PRICING LEARN Install Cloud Login

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.

Start FREE 7-day trial Install Chrome plugin

Documentation
Video Tutorials
How to
Test Sites
Forum

IT WORKS EVEN WHEN YOU SLEEP

Watch on YouTube

Use the [TABLE PLAYGROUND](#) for testing.

Test Sites

[Semantically correct tables](#)[Tables without the thead tag](#)[Tables with multiple header rows](#)

Table playground

You can train using Table selector here.

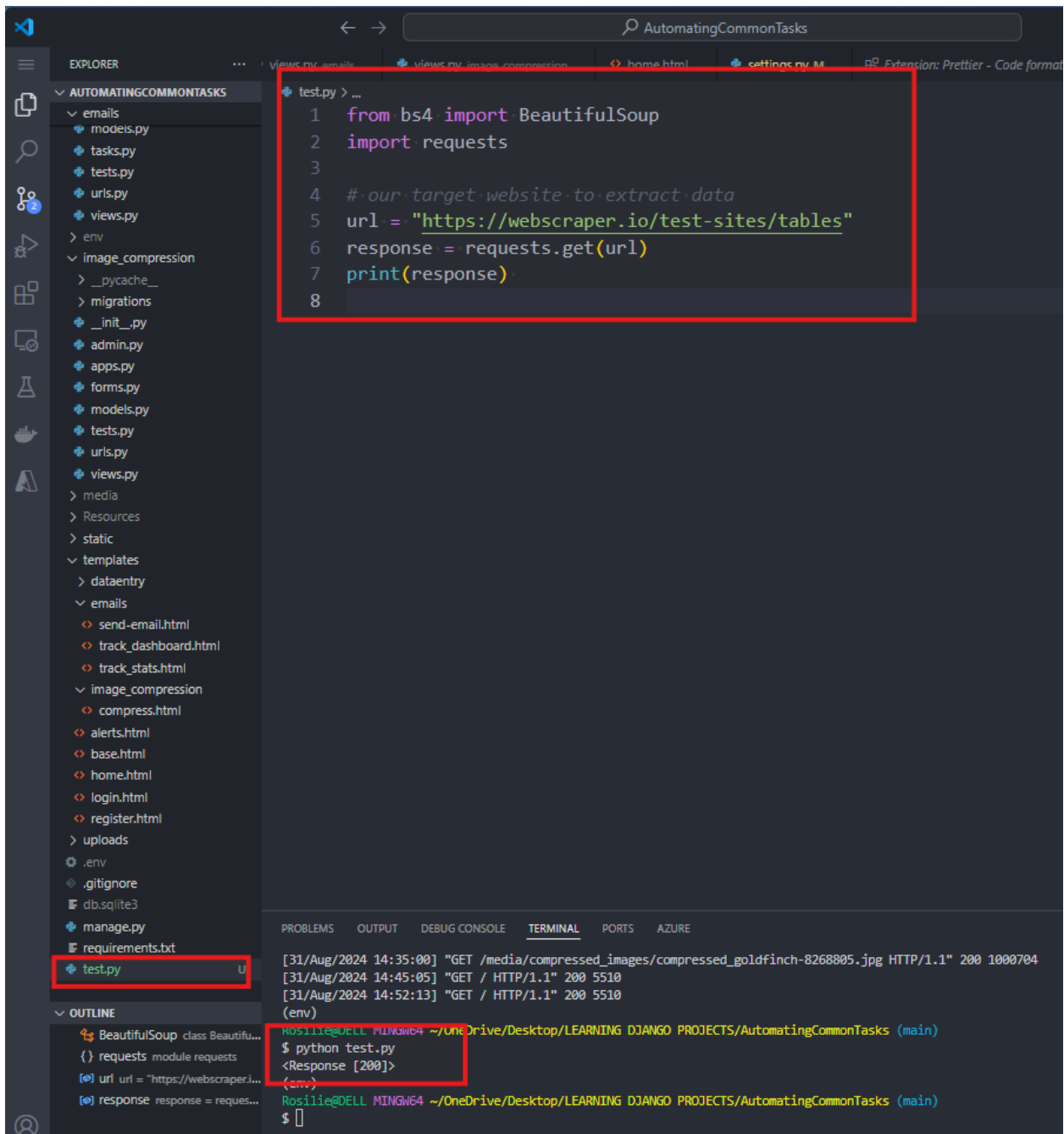
Semantically correct table with thead and tbody

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

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



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:

The screenshot shows a VS Code editor with a Python script named `test.py` open. The script is designed to scrape data from a website using `BeautifulSoup` and `requests`. The code is as follows:

```

1 from bs4 import BeautifulSoup
2 import requests
3
4 # our target website to extract data
5 url = "https://webscraper.io/test-sites/tables"
6 response = requests.get(url)
7 # print(response)
8 #
9
10 # returns the website's html code
11 print(response.content)
12
13 # returns the website's html code but in readable format
14 soup = BeautifulSoup(response.content, 'html.parser')
15 print(soup)
16

```

The Explorer sidebar on the left shows a project structure with folders like `emails`, `image_compression`, and `templates`. The main editor window displays the code for `test.py`, with a red box highlighting the line `# returns the website's html code` and the `print(response.content)` statement. The bottom status bar shows the file path: `Posilia@DELLI-MINGW64 ~/OneDrive/Desktop/I/FARNING-DJANGO-PROJECTS/AutomatingCommonTasks (main)`.

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

The screenshot shows a VS Code editor with a project named 'AutomatingCommonTasks'. The file explorer on the left shows a directory structure with files like 'test.py', 'views.py', 'urls.py', etc. The main editor displays the content of 'test.py':

```
1 from bs4 import BeautifulSoup
2 import requests
3
4 # our target website to extract data
5 url = "https://webscraper.io/test-sites/tables"
6 response = requests.get(url)
7 # print(response)
8 #
9
10 # returns the website's html code
11 print(response.content)
12
13 # returns the website's html code but in readable format
14 soup = BeautifulSoup(response.content, 'html.parser')
15 print(soup)
16
```

The terminal at the bottom shows the output of the script, which is the raw HTML of the target website, including Google Tag Manager scripts and the main content area.

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

This block contains two screenshots. The left screenshot shows a web browser at 'https://webscraper.io/test-sites/tables' with the 'Test Sites' section highlighted. The right screenshot shows the VS Code editor with the 'test.py' file modified to extract only H1 headings:

```
1 from bs4 import BeautifulSoup
2 import requests
3
4 # our target website to extract data
5 url = "https://webscraper.io/test-sites/tables"
6 response = requests.get(url)
7
8 # returns the website's html code but in readable format
9 soup = BeautifulSoup(response.content, 'html.parser')
10
11 # extracts only the headings i.e h1
12 headings1 = soup.find_all('h1')
13 print(headings1)
14
```

The terminal output shows the result of the modified script: `[<h1>Test Sites</h1>, <h1>Table playground</h1>]`.

9. To extract the images:

The image shows a VS Code editor with a file explorer on the left containing a project named 'AutomatingCommonTasks'. The main editor displays a Python script 'test.py' with the following code:

```
1 from bs4 import BeautifulSoup
2 import requests
3
4 # our target website to extract data
5 url = "https://webscraper.io/test-sites/tables"
6 response = requests.get(url)
7
8 # returns the website's html code but in readable format
9 soup = BeautifulSoup(response.content, 'html.parser')
10
11 # extracts only the headings i.e h1
12 headings1 = soup.find_all('h1')
13 headings2 = soup.find_all('h2')
14 images = soup.find_all('img')
15 print(images[0]['src']) # extracts the first image source file
16
```

The terminal at the bottom shows the execution of the script:

```
(env)
Rosllie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
[]
(env)
Rosllie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
/img/logo_white.svg
(env)
Rosllie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$
```

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

The image shows the Web Scraper website. The address bar contains the URL `https://webscraper.io/test-sites/tables`. The page features a 'Test Sites' section with a 'Table playground' area. A red box highlights a table titled 'Semantically correct table with thead and tbody'.

#	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

11. Extract only the LASTNAME OF THE SECOND TABLE:

AutomatingCommonTasks

test.py

```
6 response = requests.get(url)
7
8 # returns the website's html code but in readable format
9 soup = BeautifulSoup(response.content, 'html.parser')
10
11 # extracts only the headings i.e. h1
12 headings1 = soup.find_all('h1')
13 headings2 = soup.find_all('h2')
14 images = soup.find_all('img')[0] # gets the first image
15
16 # extracts the tables
17 table = soup.find_all('table')[1] # gets the 2nd table
18 # returns everything starting at index 1 (excludes 'th')
19 rows = table.find_all('tr')[1:]
20
21 last_names = []
22 for row in rows:
23     print(row.find_all('td')[2].get_text()) # gets the value only w/o tags
24
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

```
<td>@timbean</td>
</tr>]
(env)
Rosllie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<td>Potter</td>
<td>Snow</td>
<td>Bean</td>
<env>
Rosllie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
<td>Potter</td>
<td>Snow</td>
<td>Bean</td>
Traceback (most recent call last):
  File "C:\Users\Rosllie\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)
Rosllie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python test.py
Potter
Snow
Bean
(env)
Rosllie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$
```

OUTLINE

- BeautifulSoup class Beautiful...
- () requests module requests
- url url = "https://webscraper.i...
- response response = reques...
- soup soup = BeautifulSoup(r...
- headings1 headings1 = sou...
- headings2 headings2 = sou...

TIMELINE

The screenshot shows a VS Code editor with a project named 'AutomatingCommonTasks'. The Explorer sidebar on the left shows a file tree with folders like 'emails', 'image_compression', and 'templates', and various Python and HTML files. The main editor displays a Python script 'test.py' with the following code:

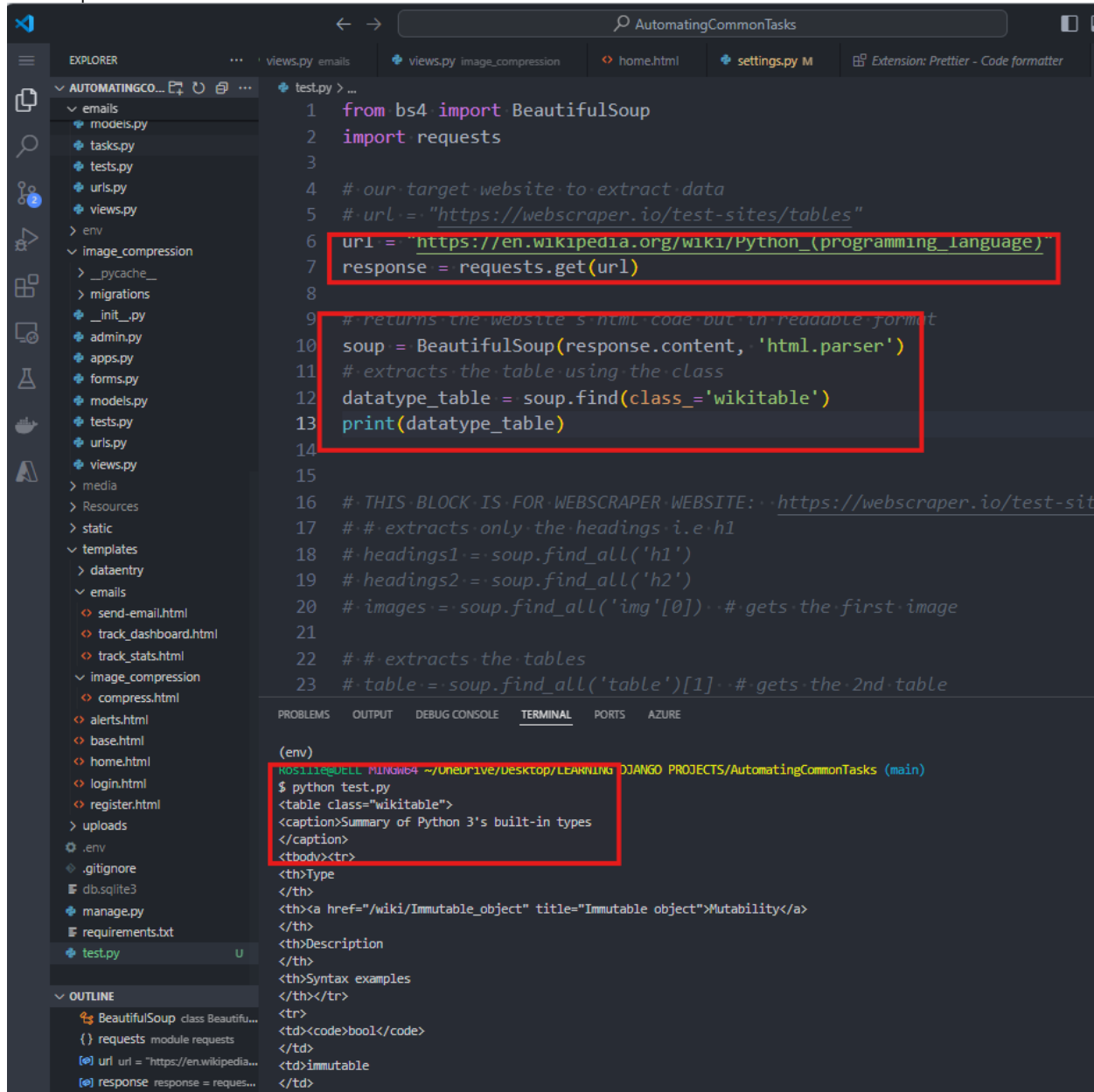
```
8 # returns the website's html code but in readable format
9 soup = BeautifulSoup(response.content, 'html.parser')
10
11 # extracts only the headings i.e h1
12 headings1 = soup.find_all('h1')
13 headings2 = soup.find_all('h2')
14 images = soup.find_all('img')[0] # gets the first image
15
16 # extracts the tables
17 table = soup.find_all('table')[1] # gets the 2nd table
18 # returns everything starting at index 1 (excludes 'th')
19 rows = table.find_all('tr')[1:]
20
21 last_names = []
22 for row in rows:
23     # adds the value only w/o tags
24     last_names.append(row.find_all('td')[2].get_text())
25 print(last_names)
```

The terminal at the bottom shows the execution of the script. It starts with a successful run, but then shows a traceback for an 'AttributeError: 'NoneType' object has no attribute 'get_text''. The final output of the script is a list of names: ['Potter', 'Snow', 'Bean'].

WEBSITE # 2: [WIKIPEDIA ON PYTHON](#)

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

We can update our TEST.PY AS:



The screenshot shows a Visual Studio Code editor with a project named 'AutomatingCommonTasks'. The Explorer sidebar on the left shows a file tree with folders like 'emails', 'image_compression', 'media', 'resources', 'static', 'templates', and 'uploads'. The main editor displays a file named 'test.py' with the following Python code:

```
1 from bs4 import BeautifulSoup
2 import requests
3
4 # our target website to extract data
5 # url = "https://webscraper.io/test-sites/tables"
6 url = "https://en.wikipedia.org/wiki/Python_(programming_language)"
7 response = requests.get(url)
8
9 # returns the website's html code but in readable format
10 soup = BeautifulSoup(response.content, 'html.parser')
11 # extracts the table using the class
12 datatype_table = soup.find(class_='wikitable')
13 print(datatype_table)
14
15
16 # THIS BLOCK IS FOR WEBSCRAPER WEBSITE: https://webscraper.io/test-sit
17 # # extracts only the headings i.e h1
18 # headings1 = soup.find_all('h1')
19 # headings2 = soup.find_all('h2')
20 # images = soup.find_all('img')[0] # gets the first image
21
22 # # extracts the tables
23 # table = soup.find_all('table')[1] # gets the 2nd table
```

The terminal output at the bottom shows the execution of the script:

```
(env)
$ 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.

red by the language, but may be used by external tools such as mypy to 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([119, 105, 115])
bytes	immutable	Sequence of bytes	b'Some ASCII' b"Some ASCII" bytes([119, 105, 115])
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)	{'key1': 1.0, 3: False}

Get citation

```

11 # extracts the table using the class
12 datatype_table = soup.find(class_='wikitable')
13 body = datatype_table.find('tbody')
14 rows = body.find_all('tr')[1:] # extracts all tr's starting at position 1
15
16 mutable_types = []
17 immutable_types = []
18
19
20 for row in rows:
21     data = row.find_all('td')
22     print(data[1]) # gets the table column # 2
23
24
25 # THIS BLOCK IS FOR WEBCRAPER WEBSITE: https://webscraper.io/test-sites/table
26 # # extracts only the headings i.e h1
27 # headings1 = soup.find_all('h1')
28 # headings2 = soup.find_all('h2')
29 # images = soup.find_all('img')[0] # gets the first image
30
31 # # extracts the tables
32 # table = soup.find_all('table')[1] # gets the 2nd table
33 # # returns everything starting at index 1 (excludes 'th')
34
35 (env)
36 NoSillie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
37 $ python test.py
38 Mutatable data types: ['bytearray\n', 'dict\n', 'list\n', 'set\n']
39
40 Immutable 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']
41 (env)

```

14. Update TEST.PY as:

```

11 # extracts the table using the class
12 datatype_table = soup.find(class_='wikitable')
13 body = datatype_table.find('tbody')
14 rows = body.find_all('tr')[1:] # extracts all tr's starting at position 1
15
16 mutable_types = []
17 immutable_types = []
18
19
20 for row in rows:
21     data = row.find_all('td')
22     if data[1].get_text() == 'mutable\n': # gets the table column # 2
23         mutable_types.append(data[0].get_text()) # adds the data type
24     else:
25         immutable_types.append(data[0].get_text())
26
27 print('Mutatable data types:', mutable_types)
28 print('=====')
29 print('Immutable data types:', immutable_types)
30
31
32 # THIS BLOCK IS FOR WEBCRAPER WEBSITE: https://webscraper.io/test-sites/table
33 # # extracts only the headings i.e h1
34
35 (env)
36 NoSillie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
37 $ python test.py
38 Mutatable data types: ['bytearray\n', 'dict\n', 'list\n', 'set\n']
39
40 Immutable 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']
41 (env)

```

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

The image shows a VS Code editor window with a project named 'AutomatingCommonTasks'. The Explorer sidebar on the left shows a file tree with folders like 'emails', 'image_compression', and 'media', and various Python and HTML files. The main editor displays a Python script 'test.py' with the following code:

```
11 # extracts the table using the class
12 datatype_table = soup.find(class_='wikitable')
13 body = datatype_table.find('tbody')
14 rows = body.find_all('tr')[1:] # extracts all tr's starting at position 1
15
16 mutable_types = []
17 immutable_types = []
18
19
20 for row in rows:
21     data = row.find_all('td')
22     if data[1].get_text() == 'mutable\n': # gets the table column # 2
23         mutable_types.append(data[0].get_text().strip()) # adds the data type
24     else:
25         immutable_types.append(data[0].get_text().strip())
26
27 print('Mutable data types:', mutable_types)
28 print('=====')
29 print('Immutable data types:', immutable_types)
30
31
32 # THIS BLOCK IS FOR WEBCRAPER WEBSITE: https://webscraper.io/test-sites/tables
33 # # extracts only the headings i.e. h1
```

The terminal window at the bottom shows the output of running the script. The first run shows mutable data types as ['bytearray\n', 'dict\n', 'list\n', 'set\n'] and immutable data types as ['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']. The second run shows the correct output with mutable types as ['bytearray', 'dict', 'list', 'set'] and immutable types as ['bool', 'bytes', 'complex', 'types.EllipsisType', 'float', 'frozenset', 'int', 'types.NoneType', 'types.NotImplementedType', 'range', 'str', 'tuple'].

16. These information are needed for STOCK MARKET ANALYSIS.