minestrone

Release 0.1

Adam Hill

Oct 22, 2022

CONTENTS

1 Behind the scenes **3**

2 Related projects **5**

2.1 Beautiful Soup related 5

2.2 Beautiful Soup replacements 5

`minestrone` is a opinionated Python library that lets you search, modify, and parse messy HTML with ease.

CHAPTER

ONE

BEHIND THE SCENES

`minestrone` utilizes `Beautiful Soup` to do all the real work but aims to provide a simple, consistent, and intuitive API to interact with an HTML document. `Beautiful Soup` provides a *lot* of functionality, although it can be hard to grok the documentation. The hope is that `minestrone` makes that functionality easier.

CHAPTER

TWO

RELATED PROJECTS

There are a few other libraries to interact with HTML in Python, but most are focused on the retrieval of HTML and searching through the document. However, they are listed below in case they might be useful.

2.1 Beautiful Soup related

- `soupy`: Soupy is a wrapper around `Beautiful Soup` that makes it easier to search through HTML and XML documents.
- `fast-soup`: fast-soup is a faster `Beautiful Soup` search via `lxml`
- `BeautifulSauce`: `Beautiful Soup`'s saucy sibling!
- `SoupCan`: SoupCan simplifies the process of designing a Python tool for extracting and displaying webpage content.

2.2 Beautiful Soup replacements

- `gazpacho`: gazpacho is a simple, fast, and modern web scraping library. The library is stable, actively maintained, and installed with zero dependencies.
- `Requests-HTML`: HTML Parsing for Humans. It intends to make parsing HTML (e.g. scraping the web) as simple and intuitive as possible.

2.2.1 Installation

To use `minestrone`, first install it using `poetry`:

```
poetry add minestrone
```

OR install it using `pip`:

```
pip install minestrone
```

2.2.2 Parsing

The `HTML` class parses a string of HTML and provides methods to *query* the DOM for specific elements.

__init__

Creates an `HTML` object from a string.

```
from minestrone import HTML
html = HTML("""
<html>
  <head>
    <title>The Dormouse's Story</title>
  </head>
  <body>
    <h1>The Dormouse's Story</h1>

    <ul>
      <li><a href="http://example.com/elsie" class="sister" id="elsie">Elsie</a></li>
      <li><a href="http://example.com/lacie" class="sister" id="lacie">Lacie</a></li>
    </ul>
  </body>
</html>
""")
```

If closing tags are missing, then they will be added as needed to make the HTML valid.

```
from minestrone import HTML
assert str(HTML("<span>dormouse")) == "<span>dormouse</span>"
```

__str__

Returns the `HTML` object as a string.

```
from minestrone import HTML
html = HTML("""
<html>
  <head>
    <title>The Dormouse's Story</title>
  </head>
  <body>
    <h1>The Dormouse's Story</h1>

    <ul>
      <li><a href="http://example.com/elsie" class="sister" id="elsie">Elsie</a></li>
      <li><a href="http://example.com/lacie" class="sister" id="lacie">Lacie</a></li>
    </ul>
  </body>
</html>
""")

assert str(html) == """<html>
```

(continues on next page)

(continued from previous page)

```
<head>
<title>The Dormouse's Story</title>
</head>
<body>
<h1>The Dormouse's Story</h1>
<ul>
<li><a href="http://example.com/elsie" class="sister" id="elsie">Elsie</a></li>
<li><a href="http://example.com/lacie" class="sister" id="lacie">Lacie</a></li>
</ul>
</body>
</html>"""
```

Note: Rendering the `HTML` into a string *will* remove preceding spaces.

2.2.3 Querying

`minestrone` allows searching through HTML via CSS selectors (similar to JQuery or other frontend libraries).

root_element

Gets the root *element* of the HTML.

```
from minestrone import HTML
html = HTML("""
<div>
  <span>Dormouse</span>
</div>
""")

assert html.root_element.name == "div"
```

query

Takes a CSS selector and returns an iterator of `Element` items.

Query by element name

```
from minestrone import HTML
html = HTML("""
<h1>The Dormouse's Story</h1>
<p>There was a table...</p>
""")

for h1 in html.query("h1"):
    assert str(h1) == "<h1>The Dormouse's Story</h1>"
```

Query by id

```
from minestrone import HTML
html = HTML("""
<ul>
  <li><a href="http://example.com/elsie" class="sister" id="elsie">Elsie</a></li>
  <li><a href="http://example.com/lacie" class="sister" id="lacie">Lacie</a></li>
</ul>
""")

for a in html.query("a#elsie"):
    assert str(a) == '<a href="http://example.com/elsie" class="sister" id="elsie">Elsie
↪</a>'
```

Query by class

```
from minestrone import HTML
html = HTML("""
<ul>
  <li><a href="http://example.com/elsie" class="sister" id="elsie">Elsie</a></li>
  <li><a href="http://example.com/lacie" class="sister" id="lacie">Lacie</a></li>
</ul>
""")

elsie_link = next(html.query("ul li a.sister"))
assert str(elsie_link) == '<a href="http://example.com/elsie" class="sister" id="elsie">
↪Elsie</a>'

lacie_link = next(html.query("ul li a.sister"))
assert str(lacie_link) == '<a href="http://example.com/lacie" class="sister" id="lacie">
↪Lacie</a>'
```

query_to_list

Exactly the same as *query* except it returns a list of `Element` items instead of a generator. This is sometimes more useful than the `query` above, but it can take more time to parse and more memory to store the data if the HTML document is large.

```
from minestrone import HTML
html = HTML("""
<ul>
  <li><a href="http://example.com/elsie" class="sister" id="elsie">Elsie</a></li>
  <li><a href="http://example.com/lacie" class="sister" id="lacie">Lacie</a></li>
</ul>
""")

assert len(html.query_to_list("a")) == 2
assert str(html.query_to_list("a")[0]) == '<a href="http://example.com/elsie" class=
↪"sister" id="elsie">Elsie</a>'
assert html.query_to_list("a") == list(html.query("a"))
```

2.2.4 Element

`Elements` are returned from *querying* methods. They have the following properties to retrieve their data.

name

Gets the name of the `Element`.

```
html = HTML("<span>Dormouse</span>")
span_element = html.root_element

assert span_element.name == "span"
```

id

Gets the id of the `Element`.

```
html = HTML("<span id='dormouse'>Dormouse</span>")
span_element = html.root_element

assert span_element.id == "dormouse"
```

attributes

Get attributes

```
html = HTML("<button class="mt-2 pb-2" disabled>Wake up</button>")
button_element = html.root_element

assert button_element.attributes == {"class": "mt-2 pb-2", "disabled": True}
```

Set attributes

```
html = HTML("<button>Go back to sleep</button>")
button_element = html.root_element
button_element.attributes = {"class": "mt-2 pb-2", "disabled": True}

assert str(button_element) == '<button class="mt-2 pb-2" disabled>Go back to sleep</
↪button>'
```

classes

Gets a list of classes for the element.

```
html = HTML("<button class="mt-2 pb-2">Wake Up</button>")
button_element = html.root_element

assert button_element.classes == ["mt-2", "pb-2"]
```

text

Get text context

```
html = HTML("<button>Wake Up</button>")
button_element = html.root_element

assert button_element.text == "Wake Up"
```

Set text content

```
html = HTML("<button>Wake Up</button>")
button_element = html.root_element

button_element.text = "Go back to sleep"

assert str(button_element) == "<button>Go back to sleep</button>"
```

2.2.5 Editing

To edit HTML, first query for an `Element` and then call one of the following methods.

prepend

Adds new text or an element **before** the calling element.

Prepend an element

```
from minestrone import HTML
html = HTML("<span>Dormouse</span>")
html.root_element.prepend(name="span", text="The", klass="mr-2")

assert str(html) == "<span class="mr-2">The</span><span>Dormouse</span>"
```

Prepend text

```
from minestrone import HTML
html = HTML("<span>Dormouse</span>")
html.root_element.prepend(text="The ")

assert html == "The <span>Dormouse</span>"
```

append

Adds text content or a new element **after** the calling element.

Append an element

```
from minestrone import HTML
html = HTML("<span>Dormouse</span>")
html.root_element.append(name="span", text="Story", klass="ml-2")

assert str(html) == "<span>Dormouse</span><span class="ml-2">Story</span>"
```

Append text

```
from minestrone import HTML
html = HTML("<span>Dormouse</span>")
html.root_element.append(text=" Story")

assert html == "<span>Dormouse</span> Story"
```