htmlBuilder Documentation

Release 1.0.0

Jaime Vinas

Mar 16, 2021

Contents

HtmlBuilder is a python library that allows you to render HTML files by writing python code. And to make use of python features, clean syntax, and object-oriented design to their full potential.

CHAPTER 1

Why should you care about this library?

When rendering HTML programmatically, there are other options available (template engines and other rendering libraries). Still, these are often limited in what they can do, or it's necessary to learn a new level of abstraction before being productive. HtmlBuilder tries to improve on this by following the next few ideas:

- **Minimal learning curve**: Users should need no more than Python and HTML knowledge to be productive using this tool.
- **Real python code**: The final code looks and behaves as you would expect from other python code.
- **Easily testable**: Users can introspect and unit test the HTML object structure **before** rendering the HTML string.

1.1 Table of content

1.1.1 Getting Started

Installation

```
pip install htmlBuilder
```

Note: Python 3.6 or later is required

A simple example

```
# import necessary tags and attributes
from htmlBuilder.tags import *
from htmlBuilder.attributes import Class, Style as InlineStyle

```

(continues on next page)

(continued from previous page)

```
# html tags are represented by classes
html = Html([],
    # any tag can receive another tag as constructor parameter
    Head([],
        Title([], "A beautiful site")
    ),
    Body([Class('btn btn-success'), InlineStyle(background_color='red', bottom='35px
↪')],
        Hr(),
        Div([],
            Div()
        )
    )
)
# no closing tags are required

# call the render() method to return tag instances as html text
print(html.render(pretty=True))
```

Output:

```
<html>
  <head>
    <title>
      A beautiful site
    </title>
  </head>
  <body class='btn btn-success' style='background-color: red; bottom: 35px'>
    <hr/>
    <div>
      <div></div>
    </div>
  </body>
</html>
```

A not so simple example

```
from htmlBuilder.attributes import Class
from htmlBuilder.tags import Html, Head, Title, Body, Nav, Div, Footer, Ul, Li

users = [ # declare data
    {
        "name": "Jose",
        "movies": ['A beautiful mind', 'Red'],
        "favorite-number": 42,
    },
    {
        "name": "Jaime",
        "movies": ['The breakfast club', 'Fight club'],
        "favorite-number": 7,
    },
    {
        "name": "Jhon",
        "movies": ['The room', 'Yes man'],
        "favorite-number": 987654321,
```

(continues on next page)

(continued from previous page)

```
    },
]

# functions can be used to handle recurring tag structures
def my_custom_nav():
    # these functions can return a tag or a list of tags ( [tag1,tag2,tag3] )
    return Nav([Class("nav pretty")],
        Div([], "A beautiful NavBar")
    )

html = Html([],
    Head([],
        Title([], "An awesome site")
    ),
    Body([],
        my_custom_nav(), # calling previously defined function
        [Div([Class(f"user-{user['name'].lower()}")],
            Div([], user['name']),
            Ul([],
                [Li([], movie) for movie in user["movies"]] # list comprehensions can be used to easily render multiple tags
            ) if user['favorite-number'] < 100 else "Favorite number is too high" # python's ternary operation is allowed too
        ) for user in users],
        Footer([], "My Footer"),
    )
)

print(html.render(pretty=True, doctype=True)) # pass doctype=True to add a document declaration
```

Output:

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      An awesome site
    </title>
  </head>
  <body>
    <nav class='nav pretty'>
      <div>
        A beautiful NavBar
      </div>
    </nav>
    <div class='user-jose'>
      <div>
        Jose
      </div>
      <ul>
        <li>
          A beautiful mind
        </li>
        <li>
          Red
        </li>
```

(continues on next page)

(continued from previous page)

```
      </ul>
    </div>
    <div class='user-jaime'>
      <div>
        Jaime
      </div>
      <ul>
        <li>
          The breakfast club
        </li>
        <li>
          Fight club
        </li>
      </ul>
    </div>
    <div class='user-jhon'>
      <div>
        Jhon
      </div>
      Favorite number is too high
    </div>
    <footer>
      My Footer
    </footer>
  </body>
</html>
```

1.1.2 HTML tags

Each HTML element type has a corresponding class defined in the `htmlBuilder.tags` module.

```
from htmlBuilder.tags import Div

html = Div()
print(html.render())
```

```
<div></div>
```

As you can see in the previous example, the `Div` instance has a `render()` method which returns a string with the resulting HTML. This is true for all HTML elements defined within `htmlBuilder.tags`.

Adding content

We'll continue by adding some text content into our HTML element.

```
from htmlBuilder.tags import Div, Span

html = Div([], "Hello world")
print(html.render())

html = Div([], Span())
print(html.render())
```

```
<div>Hello world</div>
<div><span></span></div>
```

When initializing a HTML element the first argument must be a `list`, `tuple` or iterable containing any number of `htmlBuilder.attributes.HtmlTagAttribute` instances (see *HTML attributes* for more details). Then, zero or more `htmlBuilder.tags.HtmlTag`, `str` or iterable instance objects as the element's content. That means that all of the following are valid and will render the same HTML:

Note:

All of the following examples in this page will exclude the following lines unless specified otherwise

- `from htmlBuilder.tags import *`
- `print(html.render())`

```
# All of the following will render:
# '<ul><li></li><li></li></ul>'

Ul([],
    Li(),
    Li(),
)

Ul([], [
    Li(),
    Li(),
])

Ul([],
    Li(),
    [Li()],
)

Ul([], (Li() for _ in range(2)))

#Any level of nesting is allowed when passing an iterable. So this is also valid:
Ul([], [[[[Li(), Li()]]]])
```

Adding a <!DOCTYPE> declaration

HTML documents should declare their type for browsers to know what to expect. So all HTML documents should start with a `<!DOCTYPE>` declaration.

Users can pass the `doctype=True` option to the `render()` method to handle this.

```
html = Div()
print(html.render(doctype=True, pretty=True))
```

```
<!DOCTYPE html>
<div></div>
```

For information on how to control the render process and the `pretty=True` option, see *Controlling the render process*.

1.1.3 HTML attributes

Each HTML attribute has a corresponding class defined in the `htmlBuilder.attributes` module. These can be used when initializing `htmlBuilder.tags.HtmlTag` instances.

```
from htmlBuilder.tags import Div
from htmlBuilder.attributes import Id, Class

html = Div([Id('main-div'), Class('pretty')])
print(html.render())
```

```
<div id='main-div' class='pretty'></div>
```

Every `HtmlTag` is initialized with zero or more. If it receives one or more parameters, then the first must be a python iterable with zero or more `HtmlTagAttribute` instances.

Note:

Attribute class names are created by "camel casing" their corresponding HTML names. This process is done by uppercasing ea

- `class` -> `Class`
- `id` -> `Id`
- `accept-charset` -> `AcceptCharset`
- `onkeypress` -> `Onkeypress`

Special attributes

Users can initialize all `HtmlTagAttribute` classes by passing a single `str` parameter with that attribute's value. Also, there are some frequently used attributes with more support.

`style`

This attribute can use keyword arguments to define each CSS property. Both of the following objects render the same HTML.

Note:

All of the following examples in this page will exclude the following lines unless specified otherwise

- `from htmlBuilder.tags import *`
- `from htmlBuilder.attributes import *`
- `print(html.render())`

```
# Both objects represent the same html
# <div style='color: white; background_color: black'>Hello World</div>

Div([Style(color="white", background_color="black")], "Hello World")
```

(continues on next page)

(continued from previous page)

```
Div([Style("color: white; background-color: black")], "Hello World")
```

Note: Notice that when using keyword arguments to initialize a `Style` instance, the CSS property name is constructed by replacing _ with -.

data-*

Users can handle `data-*` attributes using the `htmlBuilder.attributes.Data_` class which expects a `name` and a `value` its `__init__` first and second parameters respectively.

When rendered, the attribute name is constructed by appending the string `"data-"` with the given `name`.

```
# Both objects represent the same html
# <div data-message='Hello World'></div>

Div([Data_("message", "Hello World")])

Div([Data_(name="message", value="Hello World")])
```

1.1.4 Controlling the render process

Let's consider the following code:

```
from htmlBuilder.tags import Html, Head, Title, Body, Ul, Li
from htmlBuilder.attributes import Id

html = Html([],
    Head([],
        Title([], "My website"),
    ),
    Body([],
        Ul([Id('main-list')],
            Li([], 'Item 1'),
            Li([], 'Item 2'),
        )
    )
)

print(html.render())
```

```
<html><head><title>My website</title></head><body><ul id='main-list'><li>Item 1</li>
↪<li>Item 2</li></ul></body></html>
```

By default, `htmlBuilder` will try to keep the resulting HTML as small as possible. This is ideal for most cases and will not be a problem for browsers, but it can be for humans. So the *htmlBuilder.tasks.HtmlTag.render()* method allows you to control the output format of the rendered HTML.

Passing the option `pretty=True`:

```
print(html.render(pretty=True))
```

```
<html>
  <head>
    <title>
      My website
    </title>
  </head>
  <body>
    <ul id='main-list'>
      <li>
        Item 1
      </li>
      <li>
        Item 2
      </li>
    </ul>
  </body>
</html>
```

1.1.5 Examples

`htmlBuilder` is designed to let users apply all of their python knowledge when constructing HTML. That is why other than understanding the basics of this library, the best learning tools will be those focused on Python itself.

This page contains a list of use-cases and examples that may be useful to readers.

1.2 How to contribute?

The complete source code is available on GitHub.

Feel free to open an Issue or to submit a pull request.

1.3 License

The project is licensed under the MIT license.