Rattle.py Documentation

Release 0.0.2a1

Frodo821

Apr 20, 2019

Contents:

CHAPTER 1

Module rattlepy : Rattle.py API Reference

- **Next document:** *Elements*

Rattle.py - A Pure Python Templating Library for HTML A pure python templating library for html. Rattle.py has no special notation like Django or Jinja. For example:

```
<html>
    <head>
        <title>Hello, PTL!</title>
    </head>
    <body>
        <h1 class="heading">Hello, PTL!</h1>
    </body>
</html>
```

The above HTML equals to below Python code with rattle.py:

```
greeting = "Hello, PTL!"
with html() as html:
  with head():
    with title():
      text(greeting)
  with body():
    with node("h1", className="heading"):
      text(greeting)

# show as HTML
print(html)
```

And then, you can also make reusable components by yourself:

```
def greet(name):
  with node("div", className="greet-wrapper") as component:
    with node("h1"):
      text(f"Hello, {name}=san")
```

(continues on next page)

(continued from previous page)

```
    with node("button", className="ok-btn"):
      text("ok!")
  return component

# and using:
with greet("User"): pass
```

Enjoy!

- **Next document:** *Elements*

CHAPTER 2

Module rattlepy.elements : Rattle.py API Reference

- **Next document:** *Utils*
- **Previous document:** *Rattlepy*

HTML element short-handing functions

`rattlepy.elements.`**`a`**`(`*`**kwargs`*`)`
Create hyper-link and return it. Equivalent to `return Element("a", attributes...)`.

`rattlepy.elements.`**`article`**`(`*`**kwargs`*`)`
Create article node and return it. Equivalent to `return Element("article", attributes...)`.

`rattlepy.elements.`**`body`**`(`*`**kwargs`*`)`
Create body node and return it. Equivalent to `return Element("body", attributes...)`.

`rattlepy.elements.`**`div`**`(`*`**kwargs`*`)`
Create div node and return it. Equivalent to `return Element("div", attributes...)`.

`rattlepy.elements.`**`footer`**`(`*`**kwargs`*`)`
Create footer node and return it. Equivalent to `return Element("footer", attributes...)`.

`rattlepy.elements.`**`h1`**`(`*`**kwargs`*`)`
Create h1 node and return it. Equivalent to `return Element("h1", attributes...)`.

`rattlepy.elements.`**`h2`**`(`*`**kwargs`*`)`
Create h2 node and return it. Equivalent to `return Element("h2", attributes...)`.

`rattlepy.elements.`**`h3`**`(`*`**kwargs`*`)`
Create h3 node and return it. Equivalent to `return Element("h3", attributes...)`.

`rattlepy.elements.`**`h4`**`(`*`**kwargs`*`)`
Create h4 node and return it. Equivalent to `return Element("h4", attributes...)`.

`rattlepy.elements.`**`h5`**`(`*`**kwargs`*`)`
Create h5 node and return it. Equivalent to `return Element("h5", attributes...)`.

`rattlepy.elements.`**`h6`**`(`*`**kwargs`*`)`
Create h6 node and return it. Equivalent to `return Element("h6", attributes...)`.

rattlepy.elements.**head**(***kwargs*)
Create head node and return it. Equivalent to `return Element("head", attributes...)`.

rattlepy.elements.**header**(***kwargs*)
Create header node and return it. Equivalent to `return Element("header", attributes...)`.

rattlepy.elements.**hr**(***kwargs*)
Create hr node and return it. Equivalent to `return Element("hr", attributes...)`.

rattlepy.elements.**html**(***kwargs*)
Create html node and return it. Equivalent to `return Element("html", attributes...)`.

rattlepy.elements.**img**(***kwargs*)
Create img node and return it. Equivalent to `return Element("img", attributes...)`.

rattlepy.elements.**li**(***kwargs*)
Create li node and return it. Equivalent to `return Element("li", attributes...)`.

rattlepy.elements.**link**(***kwargs*)
Create link node and return it. Equivalent to `return Element("link", attributes...)`.

rattlepy.elements.**main**(***kwargs*)
Create main node and return it. Equivalent to `return Element("main", attributes...)`.

rattlepy.elements.**meta**(***kwargs*)
Create meta node and return it. Equivalent to `return Element("meta", attributes...)`.

rattlepy.elements.**ol**(***kwargs*)
Create ol node and return it. Equivalent to `return Element("ol", attributes...)`.

rattlepy.elements.**p**(***kwargs*)
Create paragraph node and return it. Equivalent to `return Element("p", attributes...)`.

rattlepy.elements.**script**(***kwargs*)
Create script node and return it. Equivalent to `return Element("script", attributes...)`.

rattlepy.elements.**span**(***kwargs*)
Create span node and return it. Equivalent to `return Element("span", attributes...)`.

rattlepy.elements.**style**(***kwargs*)
Create style node and return it. Equivalent to `return Element("style", attributes...)`.

rattlepy.elements.**title**(***kwargs*)
Create title node and return it. Equivalent to `return Element("title", attributes...)`.

rattlepy.elements.**ul**(***kwargs*)
Create ul node and return it. Equivalent to `return Element("ul", attributes...)`.

rattlepy.elements.**setTitle**(*string*)
Set the document title. This function is as same as

```
with title():
  text(string)
```

YOU MUST USE IN WITH EXPRESSION:

```
with setTitle("HogeHoge Page"): pass
```

- **Next document:** *Utils*
- **Previous document:** *Rattlepy*

CHAPTER 3

Module rattlepy.templating : Rattle.py API Reference

- **Next document:** *Environment*
- **Previous document:** *Utils*

Templating class and functions.

`rattlepy.templating.`**`escapeHtmlEntities`**`(`*string*`)`
Escapes certain characters.

`class` `rattlepy.templating.`**`Element`**`(`*tag*, *, *className=None*, ***kwargs*`)`
A class of an HTML element which is able to have children.

Usage:

```
with Element(tagname, attributes...):
  ...
```

For class attribute, you can use "className" instead of using "class" directly. Or, also you can use the way:

```
with Element(tagname, **{'class': 'my-class'}):
  ...
```

Attributes which are invalid identifier in Python like *data-* are also available in the way.

`exposes``(`*element=None*`)`
Changes parent element dynamically. This function aims creating custom component more easily.

Code example:

```
with Element("hoge") as hoge:
  # this element will be a child of hoge
  with Element("some-inner") as inner:
    hoge.exposes(inner)

with hoge:
  # this element will be a child of some-inner
```

(continues on next page)

(continued from previous page)

```
  with Element("other-element"):
    ...
  hoge.exposes()

with hoge:
  # this element will be a child of hoge
  with Element("some-other-element"):
    ...
```

serialize(*formatter='human_friendly'*, *force_add_doctype=False*)
Serializes HTML elements. If you want to serialize to minified form, use `str(elem)`.

formatter argument is one of ["human_friendly", "minify"]. default is "human_friendly" force_add_doctype argument is set whether force add doctype declaration even if the element is not a `<html>`

class rattlepy.templating.**SelfClosedElement**(*tag*, ***, *_outer=2*, *className=None*, ***kwargs*)
A class of an HTML element which is unable to have children like img or hr.

Usage:

```
with Element("hoge"):
  SelfClosedElement(tagname, attributes...)
```

addself(***, *outer=1*)
Add self to certain parent node.

rattlepy.templating.**text**(*content*)
This function is create text nodes. A string is expected for content argument.

Multiline contents are available in the way:

```
with Element("hoge"):
  text('''\
  |some
  |multiline
  |text''')
```

Any characters before `|` are ignored as spacers. If ending position of line spacers is not specified, all texts are inserted as text nodes.

All dangerous characters (`&` `<` `>` `"`) will be escaped. If you don't need the feature, Use `rtext` instead of this.

rattlepy.templating.**node**(*tag*, ***kwargs*)
Create Element and return it. Equivalent to `Element(tag, attributes...)`.

rattlepy.templating.**closed**(*tag*, ***kwargs*)
Create SelfClosedElement and return it. Equivalent to `SelfClosedElement(tag, attributes...)`.

- **Next document:** *Environment*
- **Previous document:** *Utils*

CHAPTER 4

Module rattlepy.utils : Rattle.py API Reference

- **Next document:** *Templating*
- **Previous document:** *Elements*

Utility functions for making html more easily.

rattlepy.utils.**createHeader**(*title*, **metas*)

Create head element. Usage:

```
with createHeader(
  "Page Title",
  {"charset": "utf-8"}):
    ...
```

This function equals to the code:

```
with head():
  for m in [{"charset": "utf-8"}]:
    meta(**m)
  setTitle("Page Title")
```

rattlepy.utils.**scaffold**(*header: rattlepy.templating.Element*)

Create html scaffold. This feature is under experimental.

- **Next document:** *Templating*
- **Previous document:** *Elements*

CHAPTER 5

Module rattlepy.environment : Rattle.py API Reference

- **Previous document:** *Templating*

Placeholder variables implementation

class rattlepy.environment.**Environment**
Holding values to replace placeholders. Threads have a separate set of variables.

THIS IS AN EXPERIMENTAL FEATURE.

Usage:

```
env = Environment()

with scaffold(createHeader("Page Title")) as html:
  with h1():
    # define a placeholder named 'title'
    text(env.define('title'))

env.title = 'Test Title'
# or
# env['title'] = 'Test Title'

print(html)

# finalizes on a certain thread.
env.dispose()
```

define(*name*)
Create a placeholder.

dispose()
Delete all data on the certain thread.

Placeholder class

class rattlepy.environment.placeholder.**Placeholder**(*parent*, *name*)
A class for placeholder in html node trees.

THIS IS AN EXPERIMENTAL FEATURE.

- **Previous document:** *Templating*

CHAPTER 6

Rattle.py API Reference

Modules:

CHAPTER 7

Indices and tables

- genindex
- modindex
- search
- *Rattle.py API Reference*

Python Module Index

r

Index

T

U