htmltag Documentation

Release 1.5

Liftoff Software

April 27, 2013

CONTENTS

Module author: Dan McDougall <daniel.mcdougall@liftoffsoftware.com>

CHAPTER

ONE

THE HTMLTAG MODULE

Note: The latest, complete documentation of htmltag can be found here: http://liftoff.github.io/htmltag/

The latest version of this module can be obtained from Github: http://liftoff.github.io/htmltag/

htmltag.py - A Python (2 *and* 3) module for wrapping whatever strings you want in HTML tags. Example:

```
>>> from htmltag import strong
>>> print(strong("SO STRONG!"))
<strong>SO STRONG!</strong>
```

What tags are supported? All of them! An important facet of modern web programming is the ability to use your own custom tags. For example:

```
>>> from htmltag import foobar
>>> foobar('Custom tag example')
'<foobar>Custom tag example</foobar>'
```

To add attributes inside your tag just pass them as keyword arguments:

```
>>> from htmltag import a
>>> print(a('awesome software', href='http://liftoffsoftware.com/'))
<a href="http://liftoffsoftware.com/">awesome software</a>
```

To work around the problem of reserved words as keyword arguments (i.e. can't have 'class="foo"') just prefix the keyword with an underscore like so:

```
>>> from htmltag import div
>>> print(div("example", _class="someclass"))
<div class="someclass">example</div>
```

Another option–which is useful for things like 'data-*' attributes–is to pass keyword arguments as a dict using the ** operator like so:

```
>>> from htmltag import li
>>> print(li("CEO", **{"class": "user", "data-name": "Dan McDougall"}))
<li class="user" data-name="Dan McDougall">CEO</li>
```

If you want to use upper-case tags just import them in caps:

```
>>> from htmltag import STRONG
>>> print(STRONG('whatever'))
<STRONG>whatever</STRONG>
```

1.1 Combining Tags and Content

You can combine multiple tags to create a larger HTML string like so:

```
>>> from htmltag import table, tr, td
>>> print(table(
...     tr(td('100'), td('200'), id="row1"),
...     tr(td('150'), td('250'), id="row2"),
... ))
<table><tr id="row1"><td>100</td><td>200</td></tr><tr id="row2"><td>150</td><td>250</td></tr></table>
```

NOTE: If you're going to do something like the above please use a *real* template language/module instead of `htmltag`. You're *probably* "doing it wrong" if you end up with something like the above in your code. For example, try Tornado's template engine.

1.2 Special Characters

Special characters that cause trouble like, '<', '>', and '&' will be automatically converted into HTML entities. If you don't want that to happen just wrap your string in `htmltag.HTML` like so:

```
>>> from htmltag import HTML, a
>>> txt = HTML("<strong>I am already HTML. Don't escape me!</strong>")
>>> a(txt, href="http://liftoffsoftware.com/")
'<a href="http://liftoffsoftware.com/"><strong>I am already HTML. Don\'t escape me!</strong></a>'
```

Since Python doesn't allow modules to have dashes (-) in their names, if you need to create a tag like that just use an underscore and change its 'tagname' attribute:

```
>>> from htmltag import foo_bar
>>> print(foo_bar('baz')) # Before
'<foo_bar>baz</foo_bar>'
>>> foo_bar.tagname = 'foo-bar'
>>> print(foo_bar('baz')) # Before
'<foo-bar>baz</foo-bar>'
```

By default self-closing HTML tags like '<img>' will not include an ending slash. To change this behavior (i.e. for XHTML) just set 'ending_slash' to `True`:

```
>>> from htmltag import img
>>> img.ending_slash = True
>>> img(src="http://somehost/images/image.png")
'<img src="http://somehost/images/image.png" />'
>>> img.ending_slash = False # Reset for later doctests
```

1.3 Protections Against Cross-Site Scripting (XSS)

By default all unsafe (XSS) content in HTML tags will be removed:

```
>>> from htmltag import a, img
>>> a(img(src="javascript:alert('pwned!')"), href="http://hacker/")
'<a href="http://hacker/">(removed)</a>'
```

If you want to change this behavior set the tag's 'safe_mode' attribute like so:

```
>>> from htmltag import a, img
>>> a.safe_mode = False
>>> img.safe_mode = False
>>> a(img(src="javascript:alert('pwned!')"), href="http://hacker/")
'<a href="http://hacker/"><img src="javascript:alert(\'pwned!\')"></a>'
>>> a.safe_mode = True # Reset for later doctests
>>> img.safe_mode = True # Ditto
```

You may also change the replacement text if you like:

```
>>> from htmltag import a, img
>>> img.replacement = "No no no!"
>>> a(img(src="javascript:alert('pwned!')"), href="http://hacker/")
'<a href="http://hacker/">No no no!</a>'
```

If you set 'replacement' to 'entities' the rejected HTML will be converted to character entities like so:

```
>>> from htmltag import a, img
>>> a.replacement = "entities"
>>> img.replacement = "entities"
>>> a(img(src="javascript:alert('pwned!')"), href="http://hacker/")
'<a href="http://hacker/">&lt;img src="javascript:alert(\'pwned!\')"&gt;</a>'
```

It is also possible to create a whitelist of allowed tags. All other tags contained therein will automatically be replaced:

```
>>> from htmltag import span
>>> whitelist = ['span', 'b', 'i', 'strong']
>>> span.whitelist = whitelist
>>> span(HTML('This is <b>bold</b> new lib is <script>awesome();</script>'))
'<span>This is <b>bold</b> new lib is (removed)awesome();(removed)</span>'
```

Lastly, all strings returned by `htmltag` are actually a subclass of `str`: `HTML`. It has a useful `escaped` property:

```
>>> from htmltag import address
>>> address.safe_mode = False # Turn off so we have a dangerous example ;)
>>> html = address('1 Hacker Ln., Nowhere, USA')
>>> print(html)
<address>1 Hacker Ln., Nowhere, USA</address>
>>> print(html.escaped)
&lt;address&gt;1 Hacker Ln., Nowhere, USA&lt;/address&gt;
```

This can be extremely useful if you want to be double-sure that no executable stuff ends up in your program's output.

CHAPTER

TWO

FUNCTIONS AND CLASSES

class `htmltag.`**`TagWrap`**(*tagname*, ***kwargs*)

Lets you wrap whatever string you want in whatever HTML tag (*tagname*) you want.

Optional Keyword Arguments:

Parameters

- **safe_mode** (*boolean*) – If `True` dangerous (XSS) content will be removed from all HTML. Defaults to `True`
- **whitelist** (*iterable*) – If given only tags that exist in the whitelist will be allowed. All else will be escaped into HTML entities.
- **replacement** (*string, "entities", or "off"*) – A string to replace unsafe HTML with. If set to "entities", will convert unsafe tags to HTML entities so they display as-is but won't be evaluated by renderers/browsers'. The defaults is "(removed)".
- **log_rejects** (*boolean*) – If `True` rejected unsafe (XSS) HTML will be logged using `logging.error()`. Defaults to `False`
- **ending_slash** (*boolean*) – If `True` self-closing HTML tags like '<img>' will not have a '/' placed before the '>'. Usually only necessary with XML and XHTML documents (as opposed to regular HTML). Defaults to `False`.

The `TagWrap` class may be used in a direct fashion (as opposed to the metaprogramming magic way: `from htmltag import sometag`):

```
>>> from htmltag import TagWrap
>>> img = TagWrap('img', ending_slash=True)
>>> print(img(src="http://company.com/someimage.png"))
<img src="http://company.com/someimage.png" />
```

The `TagWrap` class also has a `copy()` method which can be useful when you want a new tag to have the same attributes as another:

```
>>> from htmltag import TagWrap
>>> whitelist = ["b", "i", "strong", "a", "em"]
>>> replacement = "(tag not allowed)"
>>> b = TagWrap('b', whitelist=whitelist, replacement=replacement)
>>> i = b.copy('i')
>>> print(i.whitelist)
['b', 'i', 'strong', 'a', 'em']
```

Here's how you can create a number of tags with your own custom settings all at once:

```
>>> import sys
>>> from htmltag import TagWrap
```

```
>>> whitelist = ["b", "i", "strong", "a", "em"] # Whitelist ourselves
>>> replacement = "(tag not allowed)"
>>> for tag in whitelist:
...     setattr(sys.modules[__name__], tag,
...         TagWrap(tag, whitelist=whitelist, replacement=replacement))
>>> strong.replacement
'(tag not allowed)'
```

Note: `sys.modules[__name__]` is the current module; the global 'self'.

__weakref__
list of weak references to the object (if defined)

copy(*tagname*, ***kwargs*)
Returns a new instance of `TagWrap` using the given *tagname* that has all the same attributes as this instance. If *kwargs* is given they will override the attributes of the created instance.

escape(*string*)
Returns *string* with all instances of '<', '>', and '&' converted into HTML entities.

wrap(*tag*, **args*, ***kwargs*)
Returns all *args* (strings) wrapped in HTML tags like so:

```
>>> b = TagWrap('b')
>>> print(b('bold text'))
<b>bold text</b>
```

To add attributes to the tag you can pass them as keyword arguments:

```
>>> a = TagWrap('a')
>>> print(a('awesome software', href='http://liftoffsoftware.com/'))
<a href="http://liftoffsoftware.com/">awesome software</a>
```

Note: `wrap()` will automatically convert '<', '>', and '&' into HTML entities unless the wrapped string has an `__html__` method

---

CHAPTER

THREE

STRIP_XSS()

htmltag.**strip_xss**(*html*, *whitelist=None*, *replacement='(removed)'*)

This function returns a tuple containing:

- *html* with all non-whitelisted HTML tags replaced with *replacement*.
- A `set()` containing the tags that were removed.

Any tags that contain JavaScript, VBScript, or other known XSS/executable functions will also be removed.

If *whitelist* is not given the following will be used:

```
whitelist = set([
    'a', 'abbr', 'aside', 'audio', 'bdi', 'bdo', 'blockquote', 'canvas',
    'caption', 'code', 'col', 'colgroup', 'data', 'dd', 'del',
    'details', 'div', 'dl', 'dt', 'em', 'figcaption', 'figure', 'h1',
    'h2', 'h3', 'h4', 'h5', 'h6', 'hr', 'i', 'img', 'ins', 'kbd', 'li',
    'mark', 'ol', 'p', 'pre', 'q', 'rp', 'rt', 'ruby', 's', 'samp',
    'small', 'source', 'span', 'strong', 'sub', 'summary', 'sup',
    'table', 'td', 'th', 'time', 'tr', 'track', 'u', 'ul', 'var',
    'video', 'wbr'
])
```

Note: To disable the whitelisting simply set `whitelist="off"`.

Example:

```
>>> html = '<span>Hello, exploit: <img src="javascript:alert("pwned!")"></span>'
>>> html, rejects = strip_xss(html)
>>> print("'%s', Rejected: '%s'" % (html, " ".join(rejects)))
'<span>Hello, exploit: (removed)</span>', Rejected: '<img src="javascript:alert("pwned!")">'
```

Note: The default *replacement* is "(removed)".

If *replacement* is "entities" bad HTML tags will be encoded into HTML entities. This allows things like <script>'whatever'</script> to be displayed without execution (which would be much less annoying to users that were merely trying to share a code example). Here's an example:

```
>>> html = '<span>Hello, exploit: <img src="javascript:alert("pwned!")"></span>'
>>> html, rejects = strip_xss(html, replacement="entities")
>>> print(html)
<span>Hello, exploit: &lt;img src="javascript:alert("pwned!")"&gt;</span>
>>> print("Rejected: '%s'" % ", ".join(rejects))
Rejected: '<img src="javascript:alert("pwned!")">'
```

NOTE: This function should work to protect against *all* the XSS examples at OWASP. Please let us know if you find something we missed.

CHAPTER

FOUR

HTML()

class htmltag.**HTML**

New in version 1.2.0.

A subclass of Python's built-in `str` to add a simple `__html__` method that lets us know this string is HTML and does not need to be escaped. It also has an `escaped` property that will return `self` with all special characters converted into HTML entities.

__html__()

Returns `self` (we're already a string) in unmodified form.

append(**strings*)

Adds any number of supplied *strings* to `self` (we're a subclass of `str` remember) just before the last closing tag and returns a new instance of `HTML` with the result. Example:

```
>>> from htmltag import span, b
>>> html = span('Test:')
>>> print(html)
<span>Test:</span>
>>> html = html.append(' ', b('appended'))
>>> print(html)
<span>Test: <b>appended</b></span>
```

In the case of self-closing tags like '<img>' the string will simply be appended after the tag:

```
>>> from htmltag import img
>>> image = img(src="http://company.com/image.png")
>>> print(image.append("Appended string"))
<img src="http://company.com/image.png">Appended string
```

Note: Why not update ourselves in-place? Because we're a subclass of `str`; in Python strings are immutable.

---

escaped

A property that returns `self` with all characters that have special meaning (in HTML/XML) replaced with HTML entities. Example:

```
>>> print(HTML('<span>These span tags will be escaped</span>').escaped)
&lt;span&gt;These span tags will be escaped&lt;/span&gt;
```

CHAPTER

FIVE

TAGWRAP()

class htmltag.**TagWrap**(*tagname*, ***kwargs*)

Lets you wrap whatever string you want in whatever HTML tag (*tagname*) you want.

Optional Keyword Arguments:

Parameters

- **safe_mode** (*boolean*) – If `True` dangerous (XSS) content will be removed from all HTML. Defaults to `True`
- **whitelist** (*iterable*) – If given only tags that exist in the whitelist will be allowed. All else will be escaped into HTML entities.
- **replacement** (*string, "entities", or "off"*) – A string to replace unsafe HTML with. If set to "entities", will convert unsafe tags to HTML entities so they display as-is but won't be evaluated by renderers/browsers'. The defaults is "(removed)".
- **log_rejects** (*boolean*) – If `True` rejected unsafe (XSS) HTML will be logged using `logging.error()`. Defaults to `False`
- **ending_slash** (*boolean*) – If `True` self-closing HTML tags like '<img>' will not have a '/' placed before the '>'. Usually only necessary with XML and XHTML documents (as opposed to regular HTML). Defaults to `False`.

The `TagWrap` class may be used in a direct fashion (as opposed to the metaprogramming magic way: `from htmltag import sometag`):

```
>>> from htmltag import TagWrap
>>> img = TagWrap('img', ending_slash=True)
>>> print(img(src="http://company.com/someimage.png"))
<img src="http://company.com/someimage.png" />
```

The `TagWrap` class also has a `copy()` method which can be useful when you want a new tag to have the same attributes as another:

```
>>> from htmltag import TagWrap
>>> whitelist = ["b", "i", "strong", "a", "em"]
>>> replacement = "(tag not allowed)"
>>> b = TagWrap('b', whitelist=whitelist, replacement=replacement)
>>> i = b.copy('i')
>>> print(i.whitelist)
['b', 'i', 'strong', 'a', 'em']
```

Here's how you can create a number of tags with your own custom settings all at once:

```
>>> import sys
>>> from htmltag import TagWrap
```

```
>>> whitelist = ["b", "i", "strong", "a", "em"] # Whitelist ourselves
>>> replacement = "(tag not allowed)"
>>> for tag in whitelist:
...     setattr(sys.modules[__name__], tag,
...         TagWrap(tag, whitelist=whitelist, replacement=replacement))
>>> strong.replacement
'(tag not allowed)'
```

Note: `sys.modules[__name__]` is the current module; the global 'self'.

---

copy(*tagname*, ***kwargs*)
Returns a new instance of `TagWrap` using the given *tagname* that has all the same attributes as this instance. If *kwargs* is given they will override the attributes of the created instance.

escape(*string*)
Returns *string* with all instances of '<', '>', and '&' converted into HTML entities.

wrap(*tag*, **args*, ***kwargs*)
Returns all *args* (strings) wrapped in HTML tags like so:

```
>>> b = TagWrap('b')
>>> print(b('bold text'))
<b>bold text</b>
```

To add attributes to the tag you can pass them as keyword arguments:

```
>>> a = TagWrap('a')
>>> print(a('awesome software', href='http://liftoffsoftware.com/'))
<a href="http://liftoffsoftware.com/">awesome software</a>
```

Note: `wrap()` will automatically convert '<', '>', and '&' into HTML entities unless the wrapped string has an `__html__` method

---

CHAPTER

SIX

SELFWRAP()

class htmltag.**SelfWrap**(*tagname*, **args*, ***kwargs*)

This class is the magic that lets us do things like:

```
>>> from htmltag import span
```

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

h

Symbols

A

C

E

H

S

T

W