bs4 (version 4.6.0)

index
/usr/local/lib/python2.7/dist-packages/bs4/__init__.py

```
Beautiful Soup
Elixir and Tonic
"The Screen-Scraper's Friend"
http://www.crummy.com/software/BeautifulSoup/

Beautiful Soup uses a pluggable XML or HTML parser to parse a
(possibly invalid) document into a tree representation. Beautiful Soup
provides methods and Pythonic idioms that make it easy to navigate,
search, and modify the parse tree.

Beautiful Soup works with Python 2.7 and up. It works better if lxml
and/or html5lib is installed.

For more than you ever wanted to know about Beautiful Soup, see the
documentation:
http://www.crummy.com/software/BeautifulSoup/bs4/doc/
```

Package Contents

builder (package)	diagnose	testing
dammit	element	**tests** (package)

Classes

bs4.element.Tag(bs4.element.PageElement)
    BeautifulSoup

class **BeautifulSoup**(bs4.element.Tag)

```
This class defines the basic interface called by the tree builders.

These methods will be called by the parser:
  reset()
  feed(markup)

The tree builder may call these methods from its feed() implementation:
  handle_starttag(name, attrs) # See note about return value
  handle_endtag(name)
  handle_data(data) # Appends to the current data node
  endData(containerClass=NavigableString) # Ends the current data node

No matter how complicated the underlying parser is, you should be
able to build a tree using 'start tag' events, 'end tag' events,
'data' events, and "done with data" events.

If you encounter an empty-element tag (aka a self-closing tag,
like HTML's <br> tag), call handle_starttag and then
```

handle_endtag.

Method resolution order:
BeautifulSoup
bs4.element.Tag
bs4.element.PageElement
__builtin__.object

Methods defined here:

__copy__(self)

__getstate__(self)

__init__(self, markup='', features=None, builder=None, parse_only=None, from_encoding=None, exclude_encodings=None, **kwargs)
The Soup object is initialized as the 'root tag', and the provided markup (which can be a string or a file-like object) is fed into the underlying parser.

decode(self, pretty_print=False, eventual_encoding='utf-8', formatter='minimal')
Returns a string or Unicode representation of this document. To get Unicode, pass None for encoding.

endData(self, containerClass=<class 'bs4.element.NavigableString'>)

handle_data(self, data)

handle_endtag(self, name, nsprefix=None)

handle_starttag(self, name, namespace, nsprefix, attrs)
Push a start tag on to the stack.

If this method returns None, the tag was rejected by the SoupStrainer. You should proceed as if the tag had not occurred in the document. For instance, if this was a self-closing tag, don't call handle_endtag.

insert_after(self, successor)

insert_before(self, successor)

new_string(self, s, subclass=<class 'bs4.element.NavigableString'>)
Create a new NavigableString associated with this soup.

new_tag(self, name, namespace=None, nsprefix=None, **attrs)

Create a new tag associated with this soup.

object_was_parsed(self, o, parent=None, most_recent_element=None)
Add an object to the parse tree.

popTag(self)

pushTag(self, tag)

reset(self)

Data and other attributes defined here:

ASCII_SPACES = ' \n\t\x0c\r'

DEFAULT_BUILDER_FEATURES = ['html', 'fast']

NO_PARSER_SPECIFIED_WARNING = 'No parser was explicitly specified, so I\'m using...this:\n\n BeautifulSoup(YOUR_MARKUP, "%(parser)s")\n'

ROOT_TAG_NAME = u'[document]'

Methods inherited from bs4.element.Tag:

__call__(self, *args, **kwargs)
Calling a tag like a function is the same as calling its find_all() method. Eg. tag('a') returns a list of all the A tags found within this tag.

__contains__(self, x)

__delitem__(self, key)
Deleting tag[key] deletes all 'key' attributes for the tag.

__eq__(self, other)
Returns true iff this tag has the same name, the same attributes, and the same contents (recursively) as the given tag.

__getattr__(self, tag)

__getitem__(self, key)
tag[key] returns the value of the 'key' attribute for the tag, and throws an exception if it's not there.

__hash__(self)

__iter__(self)

Iterating over a tag iterates over its contents.

__len__(self)
The length of a tag is the length of its list of contents.

__ne__(self, other)
Returns true iff this tag is not identical to the other tag,
as defined in __eq__.

__nonzero__(self)
A tag is non-None even if it has no contents.

__repr__(self, encoding='unicode-escape')
Renders this tag as a string.

__setitem__(self, key, value)
Setting tag[key] sets the value of the 'key' attribute for the
tag.

__str__(self)

__unicode__(self)

childGenerator(self)
Old names for backwards compatibility

clear(self, decompose=False)
Extract all children. If decompose is True, decompose instead.

decode_contents(self, indent_level=None,
eventual_encoding='utf-8', formatter='minimal')
Renders the contents of this tag as a Unicode string.

:param indent_level: Each line of the rendering will be
indented this many spaces.

:param eventual_encoding: The tag is destined to be
encoded into this encoding. This method is _not_
responsible for performing that encoding. This information
is passed in so that it can be substituted in if the
document contains a <META> tag that mentions the document's
encoding.

:param formatter: The output formatter responsible for converting
entities to Unicode characters.

decompose(self)
Recursively destroys the contents of this tree.

encode(self, encoding='utf-8', indent_level=None,
formatter='minimal', errors='xmlcharrefreplace')

encode_contents(self, indent_level=None, encoding='utf-8', formatter='minimal')

```
Renders the contents of this tag as a bytestring.

:param indent_level: Each line of the rendering will be
   indented this many spaces.

:param eventual_encoding: The bytestring will be in this encoding.

:param formatter: The output formatter responsible for converting
   entities to Unicode characters.
```

find(self, name=None, attrs={}, recursive=True, text=None, **kwargs)

```
Return only the first child of this Tag matching the given
criteria.
```

findAll = find_all(self, name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)

```
Extracts a list of Tag objects that match the given
criteria.  You can specify the name of the Tag and any
attributes you want the Tag to have.

The value of a key-value pair in the 'attrs' map can be a
string, a list of strings, a regular expression object, or a
callable that takes a string and returns whether or not the
string matches for some custom definition of 'matches'. The
same is true of the tag name.
```

findChild = find(self, name=None, attrs={}, recursive=True, text=None, **kwargs)

```
Return only the first child of this Tag matching the given
criteria.
```

findChildren = find_all(self, name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)

```
Extracts a list of Tag objects that match the given
criteria.  You can specify the name of the Tag and any
attributes you want the Tag to have.

The value of a key-value pair in the 'attrs' map can be a
string, a list of strings, a regular expression object, or a
callable that takes a string and returns whether or not the
string matches for some custom definition of 'matches'. The
same is true of the tag name.
```

find_all(self, name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)

```
Extracts a list of Tag objects that match the given
criteria.  You can specify the name of the Tag and any
attributes you want the Tag to have.

The value of a key-value pair in the 'attrs' map can be a
string, a list of strings, a regular expression object, or a
```

callable that takes a string and returns whether or not the
string matches for some custom definition of 'matches'. The
same is true of the tag name.

get(self, key, default=None)
Returns the value of the 'key' attribute for the tag, or
the value given for 'default' if it doesn't have that
attribute.

getText = get_text(self, separator=u'', strip=False, types=(<class 'bs4.element.NavigableString'>, <class 'bs4.element.CData'>))
Get all child strings, concatenated using the given separator.

get_attribute_list(self, key, default=None)
The same as get(), but always returns a list.

get_text(self, separator=u'', strip=False, types=(<class 'bs4.element.NavigableString'>, <class 'bs4.element.CData'>))
Get all child strings, concatenated using the given separator.

has_attr(self, key)

has_key(self, key)
This was kind of misleading because has_key() (attributes)
was different from __in__ (contents). has_key() is gone in
Python 3, anyway.

index(self, element)
Find the index of a child by identity, not value. Avoids issues with
tag.contents.index(element) getting the index of equal elements.

prettify(self, encoding=None, formatter='minimal')

recursiveChildGenerator(self)

renderContents(self, encoding='utf-8', prettyPrint=False, indentLevel=0)
Old method for BS3 compatibility

select(self, selector, _candidate_generator=None, limit=None)
Perform a CSS selection operation on the current element.

select_one(self, selector)
Perform a CSS selection operation on the current element.

Data descriptors inherited from bs4.element.Tag:

children

descendants

isSelfClosing

Is this tag an empty-element tag? (aka a self-closing tag)

A tag that has contents is never an empty-element tag.

A tag that has no contents may or may not be an empty-element tag. It depends on the builder used to create the tag. If the builder has a designated list of empty-element tags, then only a tag whose name shows up in that list is considered an empty-element tag.

If the builder has no designated list of empty-element tags, then any tag with no contents is an empty-element tag.

is_empty_element

Is this tag an empty-element tag? (aka a self-closing tag)

A tag that has contents is never an empty-element tag.

A tag that has no contents may or may not be an empty-element tag. It depends on the builder used to create the tag. If the builder has a designated list of empty-element tags, then only a tag whose name shows up in that list is considered an empty-element tag.

If the builder has no designated list of empty-element tags, then any tag with no contents is an empty-element tag.

parserClass

string

Convenience property to get the single string within this tag.

:Return: If this tag has a single string child, return value is that string. If this tag has no children, or more than one child, return value is None. If this tag has one child tag, return value is the 'string' attribute of the child tag, recursively.

strings

Yield all strings of certain classes, possibly stripping them.

By default, yields only NavigableString and CData objects. So no comments, processing instructions, etc.

stripped_strings

text

Get all child strings, concatenated using the given separator.

Data and other attributes inherited from bs4.element.Tag:

quoted_colon = <_sre.SRE_Pattern object>

Methods inherited from bs4.element.PageElement:

append(self, tag)
Appends the given tag to the contents of this tag.

extract(self)
Destructively rips this element out of the tree.

fetchNextSiblings = find_next_siblings(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns the siblings of this Tag that match the given criteria and appear after this Tag in the document.

fetchParents = find_parents(self, name=None, attrs={}, limit=None, **kwargs)
Returns the parents of this Tag that match the given criteria.

fetchPrevious = find_all_previous(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns all items that match the given criteria and appear before this Tag in the document.

fetchPreviousSiblings = find_previous_siblings(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns the siblings of this Tag that match the given criteria and appear before this Tag in the document.

findAllNext = find_all_next(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns all items that match the given criteria and appear after this Tag in the document.

findAllPrevious = find_all_previous(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns all items that match the given criteria and appear before this Tag in the document.

findNext = find_next(self, name=None, attrs={}, text=None, **kwargs)
Returns the first item that matches the given criteria and appears after this Tag in the document.

findNextSibling = find_next_sibling(self, name=None, attrs={}, text=None, **kwargs)
Returns the closest sibling to this Tag that matches the given criteria and appears after this Tag in the document.

findNextSiblings = find_next_siblings(self, name=None, attrs={}, text=None, limit=None, **kwargs)

Returns the siblings of this Tag that match the given criteria and appear after this Tag in the document.

findParent = find_parent(self, name=None, attrs={}, **kwargs)
Returns the closest parent of this Tag that matches the given criteria.

findParents = find_parents(self, name=None, attrs={}, limit=None, **kwargs)
Returns the parents of this Tag that match the given criteria.

findPrevious = find_previous(self, name=None, attrs={}, text=None, **kwargs)
Returns the first item that matches the given criteria and appears before this Tag in the document.

findPreviousSibling = find_previous_sibling(self, name=None, attrs={}, text=None, **kwargs)
Returns the closest sibling to this Tag that matches the given criteria and appears before this Tag in the document.

findPreviousSiblings = find_previous_siblings(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns the siblings of this Tag that match the given criteria and appear before this Tag in the document.

find_all_next(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns all items that match the given criteria and appear after this Tag in the document.

find_all_previous(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns all items that match the given criteria and appear before this Tag in the document.

find_next(self, name=None, attrs={}, text=None, **kwargs)
Returns the first item that matches the given criteria and appears after this Tag in the document.

find_next_sibling(self, name=None, attrs={}, text=None, **kwargs)
Returns the closest sibling to this Tag that matches the given criteria and appears after this Tag in the document.

find_next_siblings(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns the siblings of this Tag that match the given criteria and appear after this Tag in the document.

find_parent(self, name=None, attrs={}, **kwargs)
Returns the closest parent of this Tag that matches the given criteria.

find_parents(self, name=None, attrs={}, limit=None, **kwargs)
Returns the parents of this Tag that match the given criteria.

find_previous(self, name=None, attrs={}, text=None, **kwargs)
Returns the first item that matches the given criteria and appears before this Tag in the document.

find_previous_sibling(self, name=None, attrs={}, text=None, **kwargs)
Returns the closest sibling to this Tag that matches the given criteria and appears before this Tag in the document.

find_previous_siblings(self, name=None, attrs={}, text=None, limit=None, **kwargs)
Returns the siblings of this Tag that match the given criteria and appear before this Tag in the document.

format_string(self, s, formatter='minimal')
Format the given string using the given formatter.

insert(self, position, new_child)

nextGenerator(self)
Old non-property versions of the generators, for backwards
compatibility with BS3.

nextSiblingGenerator(self)

parentGenerator(self)

previousGenerator(self)

previousSiblingGenerator(self)

replaceWith = replace_with(self, replace_with)

replaceWithChildren = unwrap(self)

replace_with(self, replace_with)

replace_with_children = unwrap(self)

setup(self, parent=None, previous_element=None, next_element=None, previous_sibling=None, next_sibling=None)
Sets up the initial relations between this element and

other elements.

unwrap(self)

wrap(self, wrap_inside)

Data descriptors inherited from bs4.element.PageElement:

__dict__
dictionary for instance variables (if defined)

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

next

nextSibling

next_elements

next_siblings

parents

previous

previousSibling

previous_elements

previous_siblings

Data and other attributes inherited from bs4.element.PageElement:

HTML_FORMATTERS = {None: None, 'html': <bound method type.substitute_html of <class 'bs4.element.HTMLAwareEntitySubstitution'>>, 'minimal': <bound method type.substitute_xml of <class 'bs4.element.HTMLAwareEntitySubstitution'>>}

XML_FORMATTERS = {None: None, 'html': <bound method type.substitute_html of <class 'bs4.dammit.EntitySubstitution'>>, 'minimal': <bound method type.substitute_xml of <class 'bs4.dammit.EntitySubstitution'>>}

attribselect_re = <_sre.SRE_Pattern object>

tag_name_re = <_sre.SRE_Pattern object>

Data

__all__ = ['BeautifulSoup']
__author__ = 'Leonard Richardson (leonardr@segfault.org)'
__copyright__ = 'Copyright (c) 2004-2017 Leonard Richardson'
__license__ = 'MIT'
__version__ = '4.6.0'

Author

Leonard Richardson (leonardr@segfault.org)