BeautifulSoup 4 Reference

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 provides methods and Pythonic idioms that make it easy to navigate, search, and modify the parse tree.

Beautiful Soup works with Python 2.6 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/

BeautifulSoup

class `bs4.`**`BeautifulSoup`**(*markup=''*, *features=None*, *builder=None*, *parse_only=None*, *from_encoding=None*, ***kwargs*)

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.

`decode`(*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.

handle_starttag(*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 occured in the document. For instance, if this was a self-closing tag, don't call handle_endtag.

new_string(*s*, *subclass=<class 'bs4.element.NavigableString'>*)

Create a new NavigableString associated with this soup.

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

Create a new tag associated with this soup.

object_was_parsed(*o*, *parent=None*, *most_recent_element=None*)

Add an object to the parse tree.

PageElement

class `bs4.`**PageElement**

Contains the navigational information for some part of the page (either a tag or a piece of text)

append(*tag*)

Appends the given tag to the contents of this tag.

extract()

Destructively rips this element out of the tree.

fetchNextSiblings(*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(*name=None*, *attrs={}*, *limit=None*, ***kwargs*)

Returns the parents of this Tag that match the given criteria.

fetchPrevious(*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(*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(*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(*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(*name=None*, *attrs={}*, *text=None*, ***kwargs*)

Returns the first item that matches the given criteria and appears after this Tag in the document.

findNextSibling(*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(*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(*name=None*, *attrs={}*, ***kwargs*)

Returns the closest parent of this Tag that matches the given criteria.

findParents(*name=None*, *attrs={}*, *limit=None*, ***kwargs*)

Returns the parents of this Tag that match the given criteria.

findPrevious(*name=None*, *attrs={}*, *text=None*, ***kwargs*)

Returns the first item that matches the given criteria and appears before this Tag in the document.

findPreviousSibling(*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`(*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`(*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`(*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`(*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`(*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`(*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`(*name=None*, *attrs={}*, ***kwargs*)

Returns the closest parent of this Tag that matches the given criteria.

`find_parents`(*name=None*, *attrs={}*, *limit=None*, ***kwargs*)

Returns the parents of this Tag that match the given criteria.

`find_previous`(*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`(*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(*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(*s*, *formatter='minimal'*)

Format the given string using the given formatter.

insert_after(*successor*)

Makes the given element the immediate successor of this one.

The two elements will have the same parent, and the given element will be immediately after this one.

insert_before(*predecessor*)

Makes the given element the immediate predecessor of this one.

The two elements will have the same parent, and the given element will be immediately before this one.

setup(*parent=None*, *previous_element=None*)

Sets up the initial relations between this element and other elements.

Tag

class `bs4.`**Tag**(*parser=None*, *builder=None*, *name=None*, *namespace=None*, *prefix=None*, *attrs=None*, *parent=None*, *previous=None*)

Represents a found HTML tag with its attributes and contents.

clear(*decompose=False*)

Extract all children. If decompose is True, decompose instead.

decode(*indent_level=None*, *eventual_encoding='utf-8'*, *formatter='minimal'*)

Returns a Unicode representation of this tag and its contents.

Parameters: **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.

`decode_contents`(*indent_level=None*, *eventual_encoding='utf-8'*, *formatter='minimal'*)

Renders the contents of this tag as a Unicode string.

Parameters: **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.

`decompose`()

Recursively destroys the contents of this tree.

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

Renders the contents of this tag as a bytestring.

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

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

`findAll`(*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`(*name=None*, *attrs={}*, *recursive=True*, *text=None*, ***kwargs*)

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

`findChildren`(*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(*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(*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(*separator=u''*, *strip=False*, *types=(<class 'bs4.element.NavigableString'>, <class 'bs4.element.CData'>)*)

Get all child strings, concatenated using the given separator.

get_text(*separator=u''*, *strip=False*, *types=(<class 'bs4.element.NavigableString'>, <class 'bs4.element.CData'>)*)

Get all child strings, concatenated using the given separator.

has_key(*key*)

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

index(*element*)

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

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.

`select`(*selector*, *_candidate_generator=None*)

Perform a CSS selection operation on the current element.

`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.

`text`

Get all child strings, concatenated using the given separator.