Methods:

Navigating the tree: (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-the-tree)

Going down: (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#going-down)

navigating using tag names (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-using-tag-names)

```
h_b_div_paragraphs = soup.html.body.div.p
```

Will get the `<p>` elements inside a `div` inside the `body` inside the `html` element.

`.contents` and `.children` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#contents-and-children)

```
div_children = soup.div.children
div_contents = soup.div.contents
```

This will get the direct child element(s) of the element being looked at

`.attrs` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#attributes)

```
tag.attrs
```

You can access a tag's attributes by treating the tag like a dictionary and you can access that dictionary directly as `.attrs`

`.descendants` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#descendants)

```
div_descendants = soup.div.descendants
```

This will get all the child elements of the element being looked at

`.string` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#string)

```
div_link_text = soup.div.a.string
```

If a tag has only one child, and that child is a `NavigableString` , the child is made available as `.string` , will return `'None'` if there is no string found

`.strings` and `.stripped_strings` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#strings-and-stripped-strings)

```
div_text = soup.div.strings
```

If there's more than one thing inside a tag, you can still look at just the strings. Use the `.stringsgenerator`

Going up: (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#going-up)

Top

`.parent (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#parent)`

```
title = soup.title.string.parent
```

You can access an element's parent with the `.parentattribute` . The string in the title tag has a parent, the titel tag

`.parents (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#parents)`

```
link = soup.a
for parent in link.parents:
    if parent is None:
      print parent
    else:
      print parent.name
```

You can iterate over all of an element's parents with `.parents (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#parents)` . This example uses `.parents` to travel from an `<a>` tag buried deep within the document, to the very top of the document:

Going sideways (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#going-sideways)

Top

```
.(next/previous)_(sibling/element) (http://www.crummy.com/software/BeautifulSoup/
bs4/doc/#next-sibling-and-previous-sibling)(s) (http://www.crummy.com/software/Be
autifulSoup/bs4/doc/#next-siblings-and-previous-siblings)
```

The `.(next/previous)_(sibling(s)/element(s))` can be used to navigate between page elements, getting either a single element or a list of elements. If there are no more, then these will return `'None'`

Searching the tree (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#searching-the-tree)

Top 🠅

```
.find() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#find)/
.find_all() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#fi
nd-all)/.find_...() »
(..parent(s)() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/
#find-parents-and-find-parent),
(..(next/previous)_sibling(s)() (http://www.crummy.com/software/Beaut
ifulSoup/bs4/doc/#find-next-siblings-and-find-next-sibling),
(..all_(next/previous)() (http://www.crummy.com/software/BeautifulSou
p/bs4/doc/#find-all-next-and-find-next), )
```

Returns either the first result or a list of the results

The `limit` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#the-limit-argument) argument

```
soup.find_all("a", limit=2)
```

The `recursive` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#the-recursive-argument) argument.

```
soup.find_all("a", recursive=False)\
```

Limits the number of returned results either by a number (`limit`), or to only the **direct** children (`recursive`)

Modifying the tree (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#modifying-the-tree)

Top 🠅

Changing tag names and attributes (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#changing-tag-names-and-attributes)

```
tag.name = "blockquote"
tag['class'] = 'verybold'
```

Change a tags name or attributes (attributes like they are key-value pairs)

Modifying `tag.string` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#modifying-string)

```
tag = soup.a
tag.string = "New link text."
```

Replaces the tag's contents with the string you give

```
.append() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#append)
```

It works just like calling `.append()` on a Python list

```
.new_string() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#beautifulsou
p-new-string-and-new-tag)

                                and

.new_tag() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#beautifulsoup-n
ew-string-and-new-tag)
```

You can `.append()` a new string or new tag to the document

```
.insert() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#insert)
```

Tag will be inserted at whatever numeric position you say.

`.insert_before() and .insert_after() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#insert-before-and-insert-after)`

The `.insert_before()/.insert_after()` methods insert a tag or string immediately before or after the target element

`tag.clear() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#clear)`

Removes the contents of a tag

`tag.extract() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#extract)`

Removes a tag or string from the tree. It returns the tag or string that was extracted

`tag.decompose() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#decompose)`

Removes a tag from the tree, then completely destroys it

`tag.replace_with(replacement) (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#replace_with)`

Removes a tag or string from the tree, and replaces it with the tag or string of your choice

`tag.wrap() (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#wrap)`

Wraps an element in the tag you specify and returns the new wrapper

Filters: (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#kinds-of-filters)

Top ⬆

```
def has_class_but_no_id(tag):
    return tag.has_attr('class') and not tag.has_attr('id')

soup.find_all(has_class_but_no_id)
```

The filters used inside the methods can have various formes, a sring (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#a-string), a regex (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#a-regular-expression) (`re.compile("regex")`), a list (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#a-list), `True` (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#true); which will mach everything it can, or a function (http://www.crummy.com/software/BeautifulSoup/bs4/doc/#a-function) which should return True if the right tag was found and False if not.
Here's a function that returns `True` if a tag defines the `class` attribute but doesn't define the `id` attribute:

```
def surrounded_by_strings(tag):

return (isinstance(tag.next_element, NavigableString) and \
    isinstance(tag.previous_element, NavigableString))

for tag in soup.find_all(surrounded_by_strings):
    print tag.name
```

Here's a function that returns True if a tag is surrounded by string objects:

```
soup.find('p', {'style': 'display:inline'})
```

The filters can become quite specific, here we get a p element that has a style attribute set to `'display;inline'`:

```
soup.find_all(href=re.compile("number"))
```

Or if an attribute has a certain string inside (using regex):

```
soup.find_all(class_=re.compile("ink"))

def has_six_characters(css_class):
    return css_class is not None and len(css_class) == 6

soup.find_all(class_=has_six_characters)
```

As with any keyword argument, you can pass class_ a string, a regular expression (`re.compile(regex)`), a function, or `True`