Beautiful Soup

Beautiful Soup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

Beautiful Soup

- **Beautiful Soup** is a Python library designed for quick turn-around projects like screen-scraping. Three features make it powerful:

    1. **Beautiful Soup** provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application

    2. **Beautiful Soup** automatically converts incoming documents to *Unicode* and outgoing documents to *UTF-8*. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.

    3. **Beautiful Soup** sits on top of popular Python parsers like *lxml* and *html5lib*, allowing you to try out different parsing strategies or trade speed for flexibility.

Beautiful Soup

- **Beautiful Soup** parses anything you give it, and does the tree traversal stuff for you. You can tell it "Find all the links", or "Find all the links of class externalLink", or "Find all the links whose urls match "foo.com", or "Find the table heading that's got bold text, then give me that text."

Beautiful Soup

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and
their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a>
and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc)
print(soup.prettify())
```

Beautiful Soup

Parser	Typical usage	Advantages	Disadvantages
Python's html.parser	`BeautifulSoup(markup, "html.parser")`	• Batteries included • Decent speed • Lenient (as of Python 2.7.3 and 3.2.)	• Not very lenient (before Python 2.7.3 or 3.2.2)
lxml's HTML parser	`BeautifulSoup(markup, "lxml")`	• Very fast • Lenient	• External C dependency
lxml's XML parser	`BeautifulSoup(markup, ["lxml", "xml"])` `BeautifulSoup(markup, "xml")`	• Very fast • The only currently supported XML parser	• External C dependency
html5lib	`BeautifulSoup(markup, "html5lib")`	• Extremely lenient • Parses pages the same way a web browser does • Creates valid HTML5	• Very slow • External Python dependency

Beautiful Soup

Objects — BeautifulSoup

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(open("index.html"))

soup = BeautifulSoup("<html>data</html>")
```

```
BeautifulSoup("Sacr&eacute; bleu!")
<html><head></head><body>Sacré bleu!</body></html>
```

The `BeautifulSoup` object itself represents the document as a whole. For most purposes, you can treat it as a *Tag* object. This means it supports most of the methods described in Navigating the tree and Searching the tree.

Since the `BeautifulSoup` object doesn't correspond to an actual HTML or XML tag, it has no name and no attributes. But sometimes it's useful to look at its `.name`, so it's been given the special `.name` "[document]":

Beautiful Soup

Objects — Tag

```
soup = BeautifulSoup('<b class="boldest">Extremely bold</b>')
tag = soup.b
type(tag)
# <class 'bs4.element.Tag'>
```

A `Tag` object corresponds to an XML or HTML tag in the original document

Tags have a lot of attributes and methods, and I'll cover most of them in Navigating the tree and Searching the tree. For now, the most important features of a tag are its name and attributes.

Every tag has a **name**, accessible as `.name`

```
tag.name
# u'b'
```

A tag may have any number of **attributes**. The tag `<b class="boldest">` has an attribute "class" whose value is "boldest". You can access a tag's attributes by treating the tag like a dictionary:

```
tag['class']
# u'boldest'
```

```
tag.attrs
# {u'class': u'boldest'}
```

A **string** corresponds to a bit of text within a tag. Beautiful Soup uses the `NavigableString` class to contain these bits of text:

```
tag.string
# u'Extremely bold'
type(tag.string)
# <class 'bs4.element.NavigableString'>
unicode_string = unicode(tag.string)
tag.string.replace_with("No longer bold")
```

Beautiful Soup

Navigating — Going Down

```
soup.head
# <head><title>The Dormouse's story</title></head>

soup.title
# <title>The Dormouse's story</title>

soup.body.b
# <b>The Dormouse's story</b>

soup.a
# <a class="sister" href="http://example.com/elsie"
        id="link1">Elsie</a>

soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

- `.contents`
- `.children`
- `.descendants`
- `.string`
- `.stripped_strings`

Beautiful Soup

Navigating — Going Up

- `.parent`

```
title_tag = soup.title
title_tag
# <title>The Dormouse's story</title>
title_tag.parent
# <head><title>The Dormouse's story</title></head>
```

```
title_tag.string.parent
# <title>The Dormouse's story</title>
```

```
html_tag = soup.html
type(html_tag.parent)
# <class 'bs4.BeautifulSoup'>
```

```
print(soup.parent)
# None
```

- `.parents`

```
link = soup.a
link
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</
for parent in link.parents:
    if parent is None:
        print(parent)
    else:
        print(parent.name)
# p
# body
# html
# [document]
# None
```

Beautiful Soup

Navigating — Going Sideways

```
sibling_soup = BeautifulSoup("<a><b>text1</b><c>text2</c></b></a>")
print(sibling_soup.prettify())
# <html>
#  <body>
#   <a>
#    <b>
#     text1
#    </b>
#    <c>
#     text2
#    </c>
#   </a>
#  </body>
# </html>
```

- .next_sibling
- .previous_sibling
- .next_siblings
- .previous_siblings

Beautiful Soup

Navigating — Going Back and Forth

.next_element and .previous_element

```
last_a_tag = soup.find("a", id="link3")
last_a_tag
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>

last_a_tag.next_sibling
# '; and they lived at the bottom of a well.'
```

```
last_a_tag.next_element
# u'Tillie'
```

```
last_a_tag.previous_element
# u' and\n'
last_a_tag.previous_element.next_element
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

Beautiful Soup

Searching — Filters

A string

The simplest filter is a string. Pass a string to a search method and Beautiful Soup will perform a match against that exact string. This code finds all the <b> tags in the document:

```
soup.find_all('b')
# [<b>The Dormouse's story</b>]
```

A regular expression

If you pass in a regular expression object, Beautiful Soup will filter against that regular expression using its `match()` method. This code finds all the tags whose names start with the letter "b"; in this case, the <body> tag and the <b> tag:

```
import re
for tag in
soup.find_all(re.compile("^b")):
    print(tag.name)
# body
# b
```

True

The value `True` matches everything it can. This code finds *all* the tags in the document, but none of the text strings:

```
for tag in
soup.find_all(True):
    print(tag.name)
# html
# head
# title
# body
# p
# b
# p
# a
# a
# a
# p
```

A list

If you pass in a list, Beautiful Soup will allow a string match against *any* item in that list. This code finds all the <a> tags *and* all the <b> tags:

```
soup.find_all(["a", "b"])
# [<b>The Dormouse's story</b>,
#  <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Beautiful Soup

Searching — Filters

A function

If none of the other matches work for you, define a function that takes an element as its only argument. The function should return `True` if the argument matches, and `False` otherwise.

Here's a function that returns `True` if a tag defines the "class" attribute but doesn't define the "id" attribute:

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

Pass this function into `find_all()` and you'll pick up all the <p> tags:

```
soup.find_all(has_class_but_no_id)
# [<p class="title"><b>The Dormouse's story</b></p>,
#  <p class="story">Once upon a time there were...</p>,
#  <p class="story">...</p>]
```

This function only picks up the <p> tags. It doesn't pick up the <a> tags, because those tags define both "class" and "id". It doesn't pick up tags like <html> and <title>, because those tags don't define "class".

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

```
from bs4 import NavigableString
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
# p
# a
# a
# a
# p
```

Beautiful Soup

Searching — find_all

find_all(*name*, *attrs*, *recursive*, *text*, *limit*, ***kwargs*)

```
soup.find_all("title")
# [<title>The Dormouse's story</title>]

soup.find_all("p", "title")
# [<p class="title"><b>The Dormouse's story</b></p>]

soup.find_all("a")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.find_all(id="link2")
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]

import re
soup.find(text=re.compile("sisters"))
# u'Once upon a time there were three little sisters; and their names were\n'
```

Beautiful Soup

Searching — find_all — name

```
soup.find_all("title")
# [<title>The Dormouse's story</title>]
```

Beautiful Soup

Searching — find_all — keyword

```
soup.find_all(id='link2')
# [<a class="sister" href="http://example.com/lacie"
id="link2">Lacie</a>]
```

```
soup.find_all(href=re.compile("elsie"))
# [<a class="sister" href="http://example.com/elsie"
id="link1">Elsie</a>]
```

Beautiful Soup

Searching — find_all — class

```
soup.find_all("a", class_="sister")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

```
soup.find_all(class_=re.compile("itl"))
# [<p class="title"><b>The Dormouse's story</b></p>]

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

soup.find_all(class_=has_six_characters)
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Beautiful Soup

Searching — find_all — text

```
soup.find_all(text="Elsie")
# [u'Elsie']

soup.find_all(text=["Tillie", "Elsie", "Lacie"])
# [u'Elsie', u'Lacie', u'Tillie']

soup.find_all(text=re.compile("Dormouse"))
[u"The Dormouse's story", u"The Dormouse's story"]

def is_the_only_string_within_a_tag(s):
    """Return True if this string is the only child of its parent tag."""
    return (s == s.parent.string)

soup.find_all(text=is_the_only_string_within_a_tag)
# [u"The Dormouse's story", u"The Dormouse's story", u'Elsie', u'Lacie', u'Tillie', u'...
```

Beautiful Soup

Searching — find_all — limit/recursive

The `recursive` argument

```
soup.html.find_all("title")
# [<title>The Dormouse's story</title>]

soup.html.find_all("title", recursive=False)
# []
```

The `limit` argument

```
soup.find_all("a", limit=2)
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

find(*name*, *attrs*, *recursive*, *text*, ***kwargs*)

Beautiful Soup

Searching — find_*

find_parent(*name*, *attrs*, *text*, ***kwargs*)

find_parents(*name*, *attrs*, *text*, *limit*, ***kwargs*)

find_next_sibling(*name* , *attrs*, *text*, ***kwargs*)

find_next_siblings(*name*, *attrs*, *text*, *limit*, ***kwargs*)

find_previous_sibling(*name*, *attrs*, *text*, ***kwargs*)

find_previous_siblings(*name*, *attrs*, *text*, *limit*, ***kwargs*)

find_all_next(*name*, *attrs*, *text*, *limit*, ***kwargs*)

find_next(*name*, *attrs*, *text*, ***kwargs*)

Beautiful Soup

Searching — CSS selectors

```
soup.select("title")
# [<title>The Dormouse's story</title>]

soup.select("p nth-of-type(3)")
# [<p class="story">...</p>]
```

```
soup.select(".sister")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.select("[class~=sister]")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```