tuts+

CODE > PYTHON

Scraping Webpages in Python With Beautiful Soup: The Basics

by Monty Shokeen 4 Apr 2017

Difficulty: Intermediate Length: Medium Languages: English

Python

This post is part of a series called Scraping Webpages in Python with Beautiful Soup.

Scraping Webpages in Python With Beautiful Soup: Search and DOM Modification

In a previous tutorial, I showed you how to use the Requests module to access webpages using Python. The tutorial covered a lot of topics like making GET/POST requests and downloading things like images or PDFs programmatically. The one thing missing from that tutorial was a guide on scraping webpages you accessed using Requests to extract the information that you need.

In this tutorial, you will learn about Beautiful Soup, which is a Python library to extract data from HTML files. The focus in this tutorial will be on learning the basics of the library, and more advanced topics will be covered in the next tutorial. Please note that this tutorial uses Beautiful Soup 4 for all the examples.

Installation

You can install Beautiful Soup 4 using `pip`. The package name is `beautifulsoup4`. It should work on both Python 2 and Python 3.

```
$ pip install beautifulsoup4
```

If you don't have pip installed on your system, you can directly download the Beautiful Soup 4 source tarball and install it using `setup.py`.

```
$ python setup.py install
```

BeautifulSoup is originally packaged as Python 2 code. When you install it for use with Python 3, it is automatically updated to Python 3 code. The code won't be converted unless you install the package. Here are a few common errors that you might notice:

- The "No module named HTMLParser" `ImportError` occurs when you are running the Python 2 version of the code under Python 3.
- The "No module named html.parser" `ImportError` occurs when you are running the Python 3 version of the code under Python 2.

Both the errors above can be corrected by uninstalling and reinstalling Beautiful Soup.

Installing a Parser

Before discussing the differences between different parsers that you can use with Beautiful Soup, let's write the code to create a soup.

```
from bs4 import BeautifulSoup

soup = BeautifulSoup("<html><p>This is <b>invalid HTML</p></html>", "html.pars
```

The `BeautifulSoup` object can accept two arguments. The first argument is the actual markup, and the second argument is the parser that you want to use. The different parsers are: `html.parser`, lxml, and html5lib. The `lxml` parser has two versions, an HTML parser and an XML parser.

The `html.parser` is a built-in parser, and it does not work so well in older versions of Python. You can install the other parsers using the following commands:

```
$ pip install lxml
$ pip install html5lib
```

The `lxml` parser is very fast and can be used to quickly parse given HTML. On the other hand, the `html5lib` parser is very slow, but it is also extremely lenient. Here is an example of using each of these parsers:

```
soup = BeautifulSoup("<html><p>This is <b>invalid HTML</p></html>", "html.pa
print(soup)
# <html><p>This is <b>invalid HTML</b></p></html>

soup = BeautifulSoup("<html><p>This is <b>invalid HTML</p></html>", "lxml")
print(soup)
# <html><body><p>This is <b>invalid HTML</b></p></body></html>

soup = BeautifulSoup("<html><p>This is <b>invalid HTML</p></html>", "xml")
print(soup)
# <?xml version="1.0" encoding="utf-8"?>
# <html><p>This is <b>invalid HTML</b></p></html>

soup = BeautifulSoup("<html><p>This is <b>invalid HTML</p></html>", "html5li
print(soup)
# <html><head></head><body><p>This is <b>invalid HTML</b></p></body></html>
```

The differences outlined by the above example only matter when you are parsing invalid HTML. However, most of the HTML on the web is malformed, and knowing these differences will help you in debugging some parsing errors and deciding which parser you want to use in a project. Generally, the `lxml` parser is a very good choice.

Objects in Beautiful Soup

Beautiful Soup parses the given HTML document into a tree of Python objects. There are four main Python objects that you need to know about: `Tag`, `NavigableString`, `BeautifulSoup`, and `Comment`.

The `Tag` object refers to an actual XML or HTML tag in the document. You can access the name of a tag using `tag.name`. You can also set a tag's name to something else. The name change will be visible in the markup generated by Beautiful Soup.

You can access different attributes like the class and id of a tag using `tag['class']` and `tag['id']` respectively. You can also access the whole dictionary of attributes using `tag.attrs`. You can also add, remove or modify a tag's attributes. The attributes like an element's `class` which can take multiple values are stored as a list.

The text within a tag is stored as a `NavigableString` in Beautiful Soup. It has a few useful methods like `replace_with("string")` to replace the text within a tag. You can also convert a `NavigableString` to unicode string using `unicode()`.

Beautiful Soup also allows you to access the comments in a webpage. These comments are stored as a `Comment` object, which is also basically a `NavigableString`.

You have already learned about the `BeautifulSoup` object in the previous section. It is used to represent the document as a whole. Since it is not an actual object, it does not have any name or attributes.

Getting the Title, Headings, and Links

You can extract the page title and other such data very easily using Beautiful Soup. Let's scrape the Wikipedia page about Python. First, you will have to get the markup of the page using the following code based on the Requests module tutorial to access webpages.

```
import requests
from bs4 import BeautifulSoup

req = requests.get('https://en.wikipedia.org/wiki/Python_(programming_language
soup = BeautifulSoup(req.text, "lxml")
```

Now that you have created the soup, you can get the title of the webpage using the following code:

```
soup.title
# <title>Python (programming language) - Wikipedia</title>

soup.title.name
# 'title'

soup.title.string
# 'Python (programming language) - Wikipedia'
```

You can also scrape the webpage for other information like the main heading or the first paragraph, their classes, or the `id` attribute.

```
soup.h1
# <h1 class="firstHeading" id="firstHeading" lang="en">Python (programming la

soup.h1.string
# 'Python (programming language)'

soup.h1['class']
# ['firstHeading']

soup.h1['id']
# 'firstHeading'

soup.h1.attrs
# {'class': ['firstHeading'], 'id': 'firstHeading', 'lang': 'en'}

soup.h1['class'] = 'firstHeading, mainHeading'
soup.h1.string.replace_with("Python - Programming Language")
del soup.h1['lang']
del soup.h1['id']

soup.h1
# <h1 class="firstHeading, mainHeading">Python - Programming Language</h1>
```

Similarly, you can iterate through all the links or subheading in a document using the following code:

```
for sub_heading in soup.find_all('h2'):
    print(sub_heading.text)

# all the sub-headings like Contents, History[edit]...
```

Navigating the DOM

You can navigate through the DOM tree using regular tag names. Chaining those tag names can help you navigate the tree more deeply. For example, you can get the first link in the first paragraph of the given Wikipedia page by using `soup.p.a`. All the links in the first paragraph can be accessed by using `soup.p.find_all('a')`.

You can also access all the children of a tag as a list by using `tag.contents`. To get the children at a specific index, you can use `tag.contents[index]`. You can also iterate over a tag's children by using the `.children` attribute.

Both `.children` and `.contents` are useful only when you want to access the direct or first-level descendants of a tag. To get all the descendants, you can use the `.descendants` attribute.

```
print(soup.p.contents)
# [<b>Python</b>, ' is a widely used ',.....the full list]

print(soup.p.contents[10])
# <a href="/wiki/Readability" title="Readability">readability</a>

for child in soup.p.children:
    print(child.name)
# b
# None
# a
# None
# a
# None
# ... and so on.
```

You can also access the parent of an element using the `.parent` attribute. Similarly, you can access all the ancestors of an element using the `.parents` attribute. The parent of the top-level `<html>` tag is the `BeautifulSoup` Object itself, and its parent is None.

```
print(soup.p.parent.name)
# div

for parent in soup.p.parents:
    print(parent.name)
# div
# div
# div
# body
# html
# [document]
```

You can access the previous and next sibling of an element using the `.previous_sibling` and `.next_sibling` attributes.

For two elements to be siblings, they should have the same parent. This means that the first child of an element will not have a previous sibling. Similarly, the last child of the element will not have a next sibling. In actual webpages, the previous and next siblings of an element will most probably be a new line character.

You can also iterate over all the siblings of an element using `.previous_siblings` and `.next_siblings`.

```
soup.head.next_sibling
# '\n'

soup.p.a.next_sibling
# ' for '

soup.p.a.previous_sibling
# ' is a widely used '

print(soup.p.b.previous_sibling)
# None
```

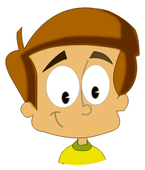
You can go to the element that comes immediately after the current element using the `.next_element` attribute. To access the element that comes immediately before the current element, use the `.previous_element` attribute.

Similarly, you can iterate over all the elements that come before and after the current element using `.previous_elements` and `.next_elements` respectively.

Final Thoughts

After completing this tutorial, you should now have a good understanding of the main differences between different HTML parsers. You should now also be able to navigate through a webpage and extract important data. This can be helpful when you want to analyze all the headings or links on a given website.

In the next part of the series, you will learn how to use the Beautiful Soup library to search and modify the DOM.



Monty Shokeen

I am a full-stack developer who also loves to write tutorials in his free time. Other than that, I love learning about new and interesting JavaScript libraries.

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by native

2 Comments **Tuts+ Hub** 1 **Login**

Recommend 2 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Deepak Asawa • 5 months ago

Good Tutorial

• Reply • Share ›

Mohamed • 10 months ago

Very good tutorial. Thanks!

• Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy

Advertisement

ENVATO TUTS+

JOIN OUR COMMUNITY

HELP

tuts+

25,385 Tutorials 1,097 Courses 20,384 Translations

Envato.com Our products Careers

Follow Envato Tuts+