



amazon appstore

FREE EBOOK: Game Jamming in Unity

Get the eBook



Advertisement

CODE > PYTHON

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

by [Monty Shokeen](#) 5 Apr 2017Difficulty: Intermediate Length: Medium Languages:

Python



This post is part of a series called [Scraping Webpages in Python with Beautiful Soup](#).

[Scraping Webpages in Python With Beautiful Soup: The Basics](#)

In the last tutorial, you learned the [basics of the Beautiful Soup library](#). Besides navigating the DOM tree, you can also search for elements with a given `class` or `id`. You can also modify the DOM tree using this library.

In this tutorial, you will learn about different methods that will help you with the search and modifications. We will be scraping the same [Wikipedia page about Python](#) from our last tutorial.

Filters for Searching the Tree

Beautiful Soup has a lot of methods for searching the DOM tree. These methods are very similar and take the same kinds of filters as arguments. Therefore, it makes sense to properly understand the different filters before reading about the methods. I will be using the same `find_all()` method to explain the difference between different filters.

The simplest filter that you can pass to any search method is a string. BeautifulSoup will then search through the document to find a tag that exactly matches the string.

```
01 | for heading in soup.find_all('h2'):
02 |     print(heading.text)
03 |
04 | # Contents
05 | # History[edit]
06 | # Features and philosophy[edit]
07 | # Syntax and semantics[edit]
08 | # Libraries[edit]
09 | # Development environments[edit]
10 | # ... and so on.
```

You can also pass a regular expression object to the `find_all()` method. This time, BeautifulSoup will filter the tree by matching all the tags against a given [regular expression](#).

```
01 | import re
02 |
03 | for heading in soup.find_all(re.compile("^h[1-6]")):
04 |     print(heading.name + ' ' + heading.text.strip())
05 |
06 | # h1 Python (programming language)
07 | # h2 Contents
08 | # h2 History[edit]
09 | # h2 Features and philosophy[edit]
10 | # h2 Syntax and semantics[edit]
11 | # h3 Indentation[edit]
12 | # h3 Statements and control flow[edit]
13 | # ... an so on.
```

The code will look for all the tags that begin with "h" and are followed by a digit from 1 to 6. In other words, it will be looking for all the heading tags in the document.

Instead of using regex, you could achieve the same result by passing a list of all the tags that you want BeautifulSoup to match against the document.

```
1 | for heading in soup.find_all(["h1", "h2", "h3", "h4", "h5", "h6"]):
2 |     print(heading.name + ' ' + heading.text.strip())
```

You can also pass `True` as a parameter to the `find_all()` method. The code will then return all the tags in the document. The output below means that there are currently 4,339 tags in the Wikipedia page that we are parsing.

```
1 | len(soup.find_all(True))
2 | # 4339
```

If you are still not able to find what you are looking for with any of the above filters, you can define your own function that takes an element as its only argument. The function also needs to return `True` if there is a match and `False` otherwise. Depending on what you need, you can make the function as complicated as it needs to be to do the job.

Here is a very simple example:

```
1 | def big_lists(tag):
2 |     return len(tag.contents) > 20 and tag.name == 'ul'
3 |
4 | len(soup.find_all(big_lists))
5 | # 13
```

The above function is going through the same Wikipedia Python page and looking for unordered lists that have more than 20 children.

Searching the DOM Tree Using Built-In Functions

One of the most popular methods for searching through the DOM is `find_all()`. It will go through all the tag's descendants and return a list of all the descendants that match your search criteria. This method has the following signature:

```
1 | find_all(name, attrs, recursive, string, limit, **kwargs)
```

The `name` argument is the name of the tag that you want this function to search for while going through the tree. You are free to provide a string, a list, a regular expression, a function, or the value `True` as a name.

You can also filter the elements in the DOM tree on the basis of different attributes like `id`, `href`, etc. You can also get all the elements with a specific attribute regardless of its value using `attribute=True`. Searching for elements with a specific class is different from searching for regular attributes. Since `class` is a reserved keyword in Python, you

will have to use the `class_` keyword argument when looking for elements with a specific class.

```
01 import re
02
03 len(soup.find_all(id=True))
04 # 425
05
06 len(soup.find_all(class_=True))
07 # 1734
08
09 len(soup.find_all(class_="mw-headline"))
10 # 20
11
12 len(soup.find_all(href=True))
13 # 1410
14
15 len(soup.find_all(href=re.compile("python")))
16 # 102
```

You can see that the document has 1,734 tags with a `class` attribute and 425 tags with an `id` attribute. If you only need the first few of these results, you can pass a number to the method as the value of `limit`. Passing this value will instruct Beautiful Soup to stop looking for more elements once it has reached a certain number. Here is an example:

```
1 soup.find_all(class_="mw-headline", limit=4)
2
3 # <span class="mw-headline" id="History">History</span>
4 # <span class="mw-headline" id="Features_and_philosophy">Features and philosophy</span>
5 # <span class="mw-headline" id="Syntax_and_semantics">Syntax and semantics</span>
6 # <span class="mw-headline" id="Indentation">Indentation</span>
```

When you use the `find_all()` method, you are telling Beautiful Soup to go through all the descendants of a given tag to find what you are looking for. Sometimes, you want to look for an element only in the direct children on a tag. This can be achieved by passing `recursive=False` to the `find_all()` method.

```
1 len(soup.html.find_all("meta"))
2 # 6
3
4 len(soup.html.find_all("meta", recursive=False))
5 # 0
6
7 len(soup.head.find_all("meta", recursive=False))
8 # 6
```

If you are interested in finding only one result for a particular search query, you can use the `find()` method to find it instead of passing `limit=1` to `find_all()`. The only difference between the results returned by these two methods is that `find_all()` returns a list with only one element and `find()` just returns the result.

```
1 soup.find_all("h2", limit=1)
2 # [<h2>Contents</h2>]
3
4 soup.find("h2")
5 # <h2>Contents</h2>
```

The `find()` and `find_all()` methods search through all the descendants of a given tag to search for an element. There are ten other very similar methods that you can use to iterate through the DOM tree in different directions.

```
01 find_parents(name, attrs, string, limit, **kwargs)
02 find_parent(name, attrs, string, **kwargs)
03
04 find_next_siblings(name, attrs, string, limit, **kwargs)
05 find_next_sibling(name, attrs, string, **kwargs)
06
07 find_previous_siblings(name, attrs, string, limit, **kwargs)
08 find_previous_sibling(name, attrs, string, **kwargs)
09
10 find_all_next(name, attrs, string, limit, **kwargs)
11 find_next(name, attrs, string, **kwargs)
12
13 find_all_previous(name, attrs, string, limit, **kwargs)
14 find_previous(name, attrs, string, **kwargs)
```

The `find_parent()` and `find_parents()` methods traverse up the DOM tree to find the given element. The `find_next_sibling()` and `find_next_siblings()` methods will iterate over all the siblings of the element that come after the current one. Similarly, the `find_previous_sibling()` and `find_previous_siblings()` methods will iterate over all the siblings of the element that come before the current one.

The `find_next()` and `find_all_next()` methods will iterate over all the tags and strings that come after the current element. Similarly, the `find_previous()` and `find_all_previous()` methods will iterate over all the tags and strings that come before the current element.

You can also search for elements using CSS selectors with the help of the `select()` method. Here are a few examples:

```
01 |
```

```

02 len(soup.select("p a"))
03 # 411
04
05 len(soup.select("p > a"))
06 # 291
07
08 soup.select("h2:nth-of-type(1)")
09 # [<h2>Contents</h2>]
10
11 len(soup.select("p > a:nth-of-type(2)"))
12 # 46
13
14 len(soup.select("p > a:nth-of-type(10)"))
15 # 6
16
17 len(soup.select("[class*=section]"))
18 # 80
19
20 len(soup.select("[class$=section]"))
21 # 20

```

Modifying the Tree

You can not only search through the DOM tree to find an element but also modify it. It is very easy to rename a tag and modify its attributes.

```

01 heading_tag = soup.select("h2:nth-of-type(2)")[0]
02
03 heading_tag.name = "h3"
04 print(heading_tag)
05 # <h3><span class="mw-headline" id="Features_and_philosophy">Feat...
06
07 heading_tag['class'] = 'headingChanged'
08 print(heading_tag)
09 # <h3 class="headingChanged"><span class="mw-headline" id="Feat...
10
11 heading_tag['id'] = 'newHeadingId'
12 print(heading_tag)
13 # <h3 class="headingChanged" id="newHeadingId"><span class="mw....
14
15 del heading_tag['id']
16 print(heading_tag)
17 # <h3 class="headingChanged"><span class="mw-headline"...

```

Continuing from our last example, you can replace a tag's contents with a given string using the `.string` attribute. If you don't want to replace the contents but add something extra at the end of the tag, you can use the `append()` method.

Similarly, if you want to insert something inside a tag at a specific location, you can use the `insert()` method. The first parameter for this method is the position or index at which you want to insert the content, and the second parameter is the content itself.

You can remove all the content inside a tag using the `clear()` method. This will just leave you with the tag itself and its attributes.

```

01 heading_tag.string = "Features and Philosophy"
02 print(heading_tag)
03 # <h3 class="headingChanged">Features and Philosophy</h3>
04
05 heading_tag.append(" [Appended This Part].")
06 print(heading_tag)
07 # <h3 class="headingChanged">Features and Philosophy [Appended This Part].</h3>
08
09 print(heading_tag.contents)
10 # ['Features and Philosophy', ' [Appended This Part].']
11
12 heading_tag.insert(1, ' Inserted this part ')
13 print(heading_tag)
14 # <h3 class="headingChanged">Features and Philosophy Inserted this part [Appended This Part].</h3>
15
16 heading_tag.clear()
17 print(heading_tag)
18 # <h3 class="headingChanged"></h3>

```

At the beginning of this section, you selected a level two heading from the document and changed it to a level three heading. Using the same selector again will now show you the next level two heading that came after the original. This makes sense because the original heading is no longer a level two heading.

The original heading can now be selected using `h3:nth-of-type(2)`. If you completely want to remove an element or tag and all the content inside it from the tree, you can use the `decompose()` method.

```

1 soup.select("h3:nth-of-type(2)") [0]
2 # <h3 class="headingChanged"></h3>
3
4 soup.select("h3:nth-of-type(3)") [0]
5 # <h3><span class="mw-headline" id="Indentation">Indentation</span>...
6
7 soup.select("h3:nth-of-type(2)") [0].decompose()
8 soup.select("h3:nth-of-type(2)") [0]
9 # <h3><span class="mw-headline" id="Indentation">Indentation</span>...

```

Once you've decomposed or removed the original heading, the heading in the third spot takes its place.

If you want to remove a tag and its contents from the tree but don't want to completely destroy the tag, you can use the `extract()` method. This method will return the tag that

it extracted. You will now have two different trees that you can parse. The root of the new tree will be the tag that you just extracted.

```
1 heading_tree = soup.select("h3:nth-of-type(2)")[0].extract()
2
3 len(heading_tree.contents)
4 # 2
```

You can also replace a tag inside the tree with something else of your choice using the `replace_with()` method. This method will return the tag or string that it replaced. It can be helpful if you want to put the replaced content somewhere else in the document.

```
01 soup.h1
02 # <h1 class="firstHeading">Python (programming language)</h1>
03
04 bold_tag = soup.new_tag("b")
05 bold_tag.string = "Python"
06
07 soup.h1.replace_with(bold_tag)
08
09 print(soup.h1)
10 # None
11 print(soup.b)
12 # <b>Python</b>
```

In the above code, the main heading of the document has been replaced with a `b` tag. The document no longer has an `h1` tag, and that is why `print(soup.h1)` now prints `None`.

Final Thoughts

After reading the two tutorials in this series, you should now be able to parse different webpages and extract important data from the document. You should also be able to retrieve the original webpage, modify it to suit your own needs, and save the modified version locally.

If you have any questions regarding this tutorial, please let me know in the comments.



Weglot

WordPress multilingual

Translate your WordPress website with Weglot. Join 20,000 users.

Advertisement



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.

[FEED](#) [LIKE](#) [FOLLOW](#) [FOLLOW](#)

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Update me weekly




Advertisement

Translations

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

[Translate this post](#)

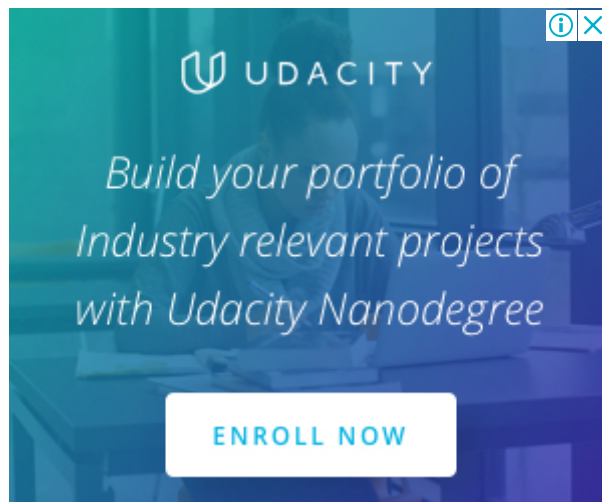
Powered by  **native**

[0 Comments](#)[Tuts+ Hub](#)[Login](#) [Recommend](#)[Share](#)[Sort by Best](#) 

LOG IN WITH

OR SIGN UP WITH DISQUS 

Be the first to comment.

[!\[\]\(7d1d6890825e83a6a4a51febe2dcc7f3_img.jpg\) Subscribe](#) [!\[\]\(5b78f4d8e2942ab203be44f938cc0a7c_img.jpg\) Add Disqus to your site](#) [Add DisqusAdd](#) [!\[\]\(1f09aec1483927ae51093bfc72ceaa0e_img.jpg\) Privacy](#)

Advertisement

[ENVATO TUTORIALS](#)[JOIN OUR COMMUNITY](#)

HELP



tuts+

25,385
Tutorials

1,097
Courses

20,384
Translations

[Envato.com](#) [Our products](#) [Careers](#)

© 2018 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

Follow Envato Tuts+