Theory: BeautifulSoup

33 minutes 0 / 5 problems solved

This topic is an introduction to `beautifulsoup`, a library that helps you analyze the HTML and the XML syntax, create parse trees and extract necessary information.

Imagine you are a student who is eager to study regional press news. If you collect every news piece manually, it will take plenty of time. `beautifulsoup` provides web scraping, a simple technique of getting data for further analysis. By combining basic operations in Python, `requests`, and `beautifulsoup` libraries, you can get all the texts and save precious time.

§1. Installation

To install `beautifulsoup`, you need a Python package manager *pip*.

```
pip install beautifulsoup4
```

Also, you need the `requests` library to send HTTP requests. It can also be installed using pip.

```
pip install requests
```

Do not forget to import these libraries before starting your work.

```
import requests

from bs4 import BeautifulSoup
```

§2. Getting started with web scraping

First, we need to create a variable that will store the content of the page. To do so, use `requests.get()` with a link as an attribute.

```
r = requests.get('https://newsineasyenglish.com/2018/05/13/air-pollution/')
```

To check whether the page was downloaded successfully, you can use `status_code`. If the returned code is *200*, there are no errors. If your code is *400* or *500*, there are some difficulties with getting the page.

```
print(r.status_code)  # 200
```

Then we use `BeautifulSoup()` class to create a parse tree of our page.

```
soup = BeautifulSoup(r.content, 'html.parser')
```

There are two parameters: `r.content` with the data of the page that is later analyzed with `html.parser`, a parser included in the standard Python library. You can install additional parsers like `lxml` via pip and use them instead of the `html.parser`. `lxml` parser is used for swift processing.

The result of the procedure is a tree. You can use the `prettify()` method to turn your tree into a nicely formatted string.standard

Current topic:

BeautifulSoup

Topic depends on:

- ✕ HTML page structure — Stage 2
- ✕ XML
- ✓ For loop 12 — Stage 1
- ✕ Requests: retrieving data

Table of contents:

```
print(soup.prettify())

# <!DOCTYPE html>
# <html lang="en-US">
#  <head>
#   <meta charset="utf-8"/>
#   <meta content="width=device-width, initial-scale=1" name="viewport"/>
#   <link href="https://gmpg.org/xfn/11" rel="profile"/>
#   <link href="https://newsineasyenglish.com/xmlrpc.php" rel="pingback"/>

#   <title>

#    Air Pollution – News in Easy English

#   </title>

#   ...
```

§3. Searching for tags

As far as you can see, the content stored in the variable `soup` is hard to follow, plus it contains a lot of unnecessary information that we do not need. Important data like texts or titles are often stored with particular tags. So, once you have decided which tags you need, there are two useful methods for finding these tags in your tree.

- `find()` method returns *the first occurrence* of the tag in the tree. This method is suitable if you are sure that your document has only one specific tag you need.

```
p1 = soup.find('title')
print(p1)  # <title>Air Pollution – News in Easy English</title>
```

- `find_all()` method returns the list of *all the results* with the tag you are searching for.

```
p2 = soup.find_all('p')
print(p2)  # [<p id="site-
description">Easy News for ESL Listening</p>, <p>
<img alt="" class="aligncenter size-full wp-image-8764" height="216" sizes="
(max-width: 382px) 100vw, ... ]
```

If the specified tags have not been found, the `find()` method returns `None` and `find_all()` returns an empty list.

We can also specify our tag using supplementary attributes: `class`, `style`, and so on... The typical structure of such specification is shown below.

```
p3 = soup.find_all('p', {'style': 'text-align: justify;'})
print(p3)  # [<p style="text-
align: justify;">The air has become very dirty in many parts of the world. This is
 of course harmful to your health. Experts say many people die from it each year.
</p>, ...]
```

We have changed our variable a little bit by adding the second parameter, a dictionary specifying the text style of elements. The keys of this dictionary are attributes of tags.

Another method connected with tag searching is `soup.<tag>` where `<tag>` is any tag you may need to find. This structure returns all the content between an opening tag and a closing tag. In the next example, the result is the content between `<head>` and `</head>`:

```
print(soup.head)
# <head>
# <meta charset="utf-8"/>
# <meta content="width=device-width, initial-scale=1" name="viewport"/>
# <link href="https://gmpg.org/xfn/11" rel="profile"/>
# ...
```

If there are several tags with the same name, the method will return only *the first occurrence*.

§4. Text and link extraction

Now we know some basics of HTML and `beautifulsoup`, it is time to extract all the necessary data. Earlier, we have learned to create a variable with lists of `<p>` tags.

```
paragraphs = soup.find_all('p', {'style': 'text-align: justify;'})
```

Now, to process all tags in this list, you can use For Loop for iteration and the `text` method helps to get text data.

```
for p in paragraphs:
    print(p.text + '\n')
```

Each `p.text` returns *a text paragraph* from the page.

There is another helpful method that can be used to get tag attributes. Let's get *hyperlinks* while working with link tags. After collecting hyperlinks, we can use them for further requests and collecting related data. The developers of `beautifulsoup` also allow users to extract links using the `get()` method. Just write down a quoted attribute of the tag you need to extract in round brackets.

```
a = soup.find_all('a')
for i in a:
    print(i.get('href'))  # https://newsineasyenglish.com/ ...
```

§5. Summary

In this topic, we have learned the main operations with `beautifulsoup`:

- How to create a parse tree,
- How to search for tags,
- How to extract texts and links.

We have also got acquainted with the basics of HTML. `beautifulsoup` seems to be elaborate at first, but after some practicing, you will get used to this library and soon you will be able to collect rich varieties of data. Find more on `beautifulsoup` in the official Beautiful Soup Documentation.

Report a typo

93 users liked this theory. **3** didn't like it. **What about you?**

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion