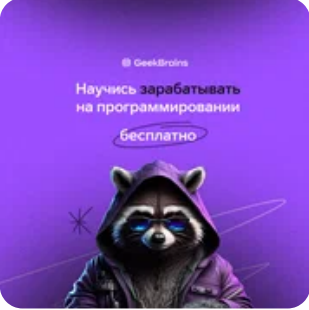


ХОЧУ ПОМОЧЬ
ПРОЕКТУ

Модуль BeautifulSoup4 в Python, разбор HTML



gb.ru

Python: бесплатное обучение! 3 курса по программированию

Научись писать код на Python с нуля на 3 бесплатных онлайн-курсах! 0 руб. 3 курса

Узнать больше

skillfactory.ru

Курс по пентесту -
от 4 200 р/мес

от 4 200 ₽ -40% 7 000 ₽

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

Рассрочка

Преподаватели-практики

Возврат денег

Узнать больше

Модуль BeautifulSoup4 в Python, разбор HTML

Документов HTML и XML

Библиотека Python для извлечения данных из файлов HTML и XML. Для естественной навигации, модуль BeautifulSoup4, по умолчанию использует встроенный в Python парсер [html.parser](#). Среди других парсеров Python, таких как lxml, html5lib и xml (для разбора XML-документов).

Установка BeautifulSoup4 в виртуальное окружение:

если нет

VirtualEnv

создания виртуального окружения

```
pip install -U beautifulsoup4
```

Содержание:

- [Выбор парсера для использования в BeautifulSoup4.](#)
 - [Парсер lxml.](#)
 - [Парсер html5lib.](#)
 - [Встроенный в Python парсер html.parser.](#)
- [Основные приемы работы с BeautifulSoup4.](#)
 - [Навигация по структуре HTML-документа.](#)
 - [Извлечение URL-адресов.](#)
 - [Извлечение текста HTML-страницы.](#)
 - [Поиск тегов по HTML-документу.](#)
 - [Поиск тегов при помощи CSS селекторов.](#)
 - [Дочерние элементы.](#)
 - [Родительские элементы.](#)
 - [Изменение имен тегов HTML-документа.](#)
 - [Добавление новых тегов в HTML-документ.](#)
 - [Удаление и замена тегов в HTML-документе.](#)
 - [Изменение атрибутов тегов HTML-документа.](#)

Выбор парсера для использования в BeautifulSoup4.

BeautifulSoup4 представляет один интерфейс для разных парсеров, но парсеры неодинаковы. Разные парсеры, анализируя один и того же документ создадут различные деревья HTML. Самые большие различия будут между парсерами HTML и XML. Так же парсеры различаются скоростью разбора HTML документа.

Если дать BeautifulSoup4 идеально оформленный документ HTML, то различий построенного HTML-дерева не будет. Один парсер будет быстрее другого, но все они будут давать структуру, которая выглядит точно так же, как оригинальный документ HTML. Но если документ оформлен с ошибками, то различные парсеры дадут разные результаты.

Различия в построении HTML-дерева разными парсерами, разберем на короткой HTML-разметке: <a></p>.

Парсер lxml.

Характеристики:

- Вверх
- Для запуска примера, необходимо установить модуль lxml.
 - Очень быстрый, имеет внешнюю зависимость от языка C.

- Нестрогий.

```
>>> from bs4 import BeautifulSoup
>>> BeautifulSoup("<a></p>", "lxml")
# <a></p></a>
```



РЕКЛАМА • 16+

skillfactory.ru

Курс по пентесту - от 4 200 р/мес

от 4 200 ₪ −40% 7 000 ₪

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

Рассрочка

Преподаватели-практики

Возврат денег

Узнать больше

Включен в теги <body> и <html>, а висячий тег </p> просто игнорируется.

Можно установить модуль html5lib.

Как это делает браузер, создавая валидный HTML5.

```
>>> from bs4 import BeautifulSoup
>>> BeautifulSoup("<a></p>", "html5lib")
# <a></p></a></body></html>
```

html5lib НЕ игнорирует висячий тег </p>, и к тому же добавляет открывающий тег <p>. Также добавляет <head> (lxml этого не сделал).

Парсер html.parser.

Медленнее, чем lxml, но установка проще.

Парсер html5lib не такой быстрый, как lxml.

```
>>> from bs4 import BeautifulSoup
>>> BeautifulSoup("<a></p>", 'html.parser')
# <a></a>
```

Как и lxml, встроенный в Python парсер игнорирует закрывающий тег </p>. В отличие от html5lib, этот парсер не делает попытки создать правильно оформленный HTML-документ, добавив теги <html> или <body>.

Вывод: Парсер html5lib использует способы, которые являются частью стандарта HTML5, поэтому он может претендовать на то, что его подход самый "правильный".

Основные приемы работы с BeautifulSoup4.

Чтобы разобрать HTML-документ, необходимо передать его в конструктор класса BeautifulSoup(). Можно передать строку или открытый дескриптор файла:

```
from bs4 import BeautifulSoup

# передаем объект открытого файла
with open("index.html") as fp:
    soup = BeautifulSoup(fp, 'html.parser')

# передаем строку
soup = BeautifulSoup("<html>a web page</html>", 'html.parser')
```

Первым делом документ конвертируется в Unicode, а HTML-мнемоники конвертируются в символы Unicode:

```
>>> from bs4 import BeautifulSoup
>>> html = "<html><head></head><body>Sacré; bleu!</body></html>"
>>> parse = BeautifulSoup(html, 'html.parser')
>>> print(parse)
# <html><head></head><body>Sacré bleu!</body></html>
```

Дальнейшие примеры будут разбираться на следующей HTML-разметке.

```
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
```

Elsie,
Lacie and
Tillie;
and they lived at the bottom of a well.</p>



конструктор класса BeautifulSoup() создает объект, который представляет документ в виде

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'html.parser')
```

Once upon a time there were three little sisters; and their names were


```
#
#
# <a class="sister" href="http://example.com/lacie" id="link2">
#   Lacie
# </a>
# and
# <a class="sister" href="http://example.com/tillie" id="link3">
#   Tillie
# </a>
# ; and they lived at the bottom of a well.
# </p>
# <p class="story">
#   ...
# </p>
# </body>
# </html>
```

Навигация по структуре HTML-документа:

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

# извлечение имя тега
>>> soup.title.name
# 'title'

# извлечение текста тега
>>> soup.title.string
# 'The Dormouse's story'


# извлечение первого тега `<p>`
>>> soup.p
# <p class="title"><b>The Dormouse's story</b></p>

# извлечение второго тега `<p>` и
# представление его содержимого списком
>>> soup.find_all('p')[1].contents
# ['Once upon a time there were three little sisters; and their names were\n',
# <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>;
# and they lived at the bottom of a well.</p>]
```

Вверх

```
# ' and\n',
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>,
# ';\nand they lived at the bottom of a well.']

"
```



РЕКЛАМА • 16+

skillfactory.ru

Курс по пентесту - от 4 200 р/мес

от 4 200 ₪ -40% 7 000 ₪

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

Рассрочка

Преподаватели-практики

Возврат денег

Узнать больше

```
vide генератора
;s
ings at 0x7ffa2eb43ac0>

можно при помощи атрибутов .previous_sibling и .next_sibling. Например, в представленном
ег <p> - следовательно они находятся на одном уровне.

'/example.com/elsie" id="link1">Elsie</a>

three little sisters; and their names were\n'

'/example.com/lacie" id="link2">Lacie</a>

евые элементы данного тега с помощью .next_siblings или .previous_siblings.

ings:

'/example.com/lacie" id="link2">Lacie</a>

'/example.com/tillie" id="link3">Tillie</a>

# ';\nand they lived at the bottom of a well.'
```

```
for sibling in soup.find(id="link3").previous_siblings:
    print(repr(sibling))
# ' and\n'
# <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
# ',\n'
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
# 'Once upon a time there were three little sisters; and their names were\n'
```

Атрибут `.next_element` строки или HTML-тега указывает на то, что было разобрано непосредственно после него. Это могло бы быть тем же, что и `.next_sibling`, но обычно результат резко отличается.

Возьмем последний тег `<a>`, его `.next_sibling` является строкой: конец предложения, которое было прервано началом тега `<a>`:

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

last_a.next_sibling
# ';\nand they lived at the bottom of a well.'
```

Однако `.next_element` этого тега `<a>` - это то, что было разобрано сразу после тега `<a>` - это слово *Tillie*, а не остальная часть предложения.

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

Это потому, что в оригинальной разметке слово *Tillie* появилось перед точкой с запятой. Парсер обнаружил тег `<a>`, затем слово *Tillie*, затем закрывающий тег ``, затем точку с запятой и оставшуюся часть предложения. Точка с запятой находится на том же уровне, что и тег `<a>`, но слово *Tillie* встретилось первым.

Атрибут `.previous_element` является полной противоположностью `.next_element`. Он указывает на элемент, который был обнаружен при разборе непосредственно перед текущим:

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

```
for element in last_a_tag.next_elements:
```



Узнать больше

это извлечение URL-адресов, найденных на странице в HTML-тегах <a>:

-страницы.

извлечь весь текст со HTML-страницы:

делителями '\n'

```
True)
```

```
mouse's story\n
```

```
# once upon a time there were three little sisters; and their names were\n# Elsie\n,\nLacie\nand\nTillie\n;\nand they lived at the bottom of a well.\n..."
```

а можно создать список строк, а потом форматировать как надо

```
>>> [text for text in soup.stripped_strings]
```

```
# ["The Dormouse's story",
```

"The Dormouse's story",

```
# 'Once upon a time there were three little sisters; and their names were',
```

```
# 'Elsie',
```

' ' , ,

```
# 'Lacie',
```

'and',

'Tillie',

';\nand they lived at the bottom of a well.',

'...']

Найти первый совпавший HTML-тег можно методом BeautifulSoup.find(), а всех совпавших элементов - BeautifulSoup.find_all().

```
# ищет все теги '<title>'
```

```
>>> soup.find_all("title")
```

```
# [<title>The Dormouse's story</title>]
```

ищет все теги `` и все теги `**`**

```
>>> soup.find_all(["a", "b"])
```

[**The Dormouse's story**,

```
# <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>]
```

```
# ищет все теги `<p>` с CSS классом "title"
```

```
>>> soup.find_all("p", "title")
```

```
# [

<b>The Dormouse's story</b></p>]


```

```
# ищет все теги с CSS классом, в именах которых встречается "itl"
```

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


```
# [

<b>The Dormouse's story</b></p>]


```

ДВЕРХе теги с `id="link2"`

РЕКЛАМА · 16+



skillfactory.ru

Курс по пентесту - от 4 200 р/мес

от 4 200 ₹

-40%

 7 000 ₹

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

Рассрочка >

Преподаватели-практики >

Возврат денег >

Узнать больше

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

# ищет все теги <a>, содержащие указанные атрибуты
# class: 'sister', 'id': 'link1'})
# href="http://example.com/elsie" id="link1">Elsie</a>]

# который содержит 'sisters'
# "sisters"))
# three little sisters; and their names were\n'

# начинаются на букву 'b'
# re.compile("^b")):

# не текстовые строки

" a
# a
# p
```

Поиск тегов при помощи CSS селекторов:

```
>>> soup.select("title")
# [<title>The Dormouse's story</title>]
>>> soup.select("p:nth-of-type(3)")
# [<p class="story">...</p>]
```

Поиск тега под другими тегами:

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

Поиск тега непосредственно под другими тегами:

```
>>> soup.select("head > title")
# [<title>The Dormouse's story</title>]
>>> soup.select("p > a:nth-of-type(2)")
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
>>> soup.select("p > #link1")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>]
```

Поиск одноуровневых элементов:

```
# поиск всех '.sister' в которых нет '#link1'
>>> soup.select("#link1 ~ .sister")
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
# поиск всех '.sister' в которых есть '#link1'
>>> soup.select("#link1 + .sister")
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
# поиск всех <a> у которых есть сосед <p>
```

Вверх

Поиск тега по классу CSS:

```
>>> 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>]
```



 skillfactory.ru

Курс по пентесту - от 4 200 р/мес

от 4 200 ₪ -40% 7 000 ₪

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

- Рассрочка >
- Преподаватели-практики >
- Возврат денег >

Узнать больше

```
//example.com/elsie" id="link1">Elsie</a>]
//example.com/lacie" id="link2">Lacie</a>]
```

оних элементов тега. Если посмотреть на HTML-разметку в коде ниже, то, непосредственными > будут являться три тега и тег со всеми вложенными тегами.

оды строк \n и пробелы между тегами, так же будут считаться дочерними элементами. Так что одный HTML к "нормальному виду", например так: re.sub(r'>\s+<', '><', html.replace('\n',

```
        </ul>
        <li>текст 3</li>
    </ul>
</div>
"""

>>> from bs4 import BeautifulSoup
>>> root = BeautifulSoup(html, 'html.parser')
# найдем в дереве первый тег `<ul>`
>>> first_ul = root.ul
# извлекаем список непосредственных дочерних элементов
# переводы строк `\\n` и пробелы между тегами так же
# распознаются как дочерние элементы
>>> first_ul.contents
# ['\\n', <li>текст 1</li>, '\\n', <li>текст 2</li>, '\\n', <ul>
# <li>текст 2-1</li>
# <li>текст 2-2</li>
# </ul>, '\\n', <li>текст 3</li>, '\\n']

# убираем переводы строк `\\n` как из списка, так и из тегов
# лучше конечно сразу убрать переводы строк из исходного HTML
>>> [str(i).replace('\\n', '') for i in first_ul.contents if str(i) != '\\n']
# ['<li>текст 1</li>',
#  '<li>текст 2</li>',
#  '<ul><li>текст 2-1</li><li>текст 2-2</li></ul>',
#  '<li>текст 3</li>']

# то же самое, что и `first_ul.contents`
# только в виде итератора
>>> first_ul.children
# <list_iterator object at 0x7ffa2eb52460>
```

Извлечение ВСЕХ дочерних элементов. Эта операция похожа на рекурсивный обход HTML-дерева в глубину от выбранного тега.


```
>>> import re
# сразу уберем переводы строк из исходного HTML
>>> html = re.sub(r'>\s+<', '><', html.replace('\\n', ''))
>>> root = BeautifulSoup(html, 'html.parser')
# найдем в дереве первый тег `<ul>`
>>> first_ul = root.ul
# 

Вверх

зем список ВСЕХ дочерних элементов
>>> list(first_ul.descendants)
```

```
# [<li>текст 1</li>,  
# 'текст 1',  
# <li>текст 2</li>,  
# 'текст 2',  
# 'текст 3',  
# <ul><li>текст 2-1</li><li>текст 2-2</li></ul>,&br/># <li>текст 3</li>]
```



 skillfactory.ru

Курс по пентесту - от 4 200 р/мес

от 4 200 ₪ −40% 7 000 ₪

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

- Рассрочка >
- Преподаватели-практики >
- Возврат денег >

Узнать больше

текст, который находится внутри тега, так же считается дочерним элементом этого тега.

то дочернего элемента (как в примерен выше) и необходимо извлечь только текст, то можно использовать генератор `.stripped_strings`.

полностью удаляет все переводы строк `\n` и пробелы между тегами в исходном HTML-документе.

```
'текст 2-1', 'текст 2-2', 'текст 3']
```

```
stripped_strings at 0x7ffa2eb43ac0>  
(<li>текст 2-1</li>,<br> <li>текст 2-2</li>,<br> <li>текст 3</li>)
```

родительскому элементу, необходимо использовать атрибут `.parent`.

```
html = """  
<div>  
  <ul>  
    <li>текст 1</li>  
    <li>текст 2</li>  
    <ul>  
      <li>текст 2-1</li>  
      <li>текст 2-2</li>  
    </ul>  
    <li>текст 3</li>  
  </ul>  
</div>  
"""  
  
>>> from bs4 import BeautifulSoup  
>>> import re  
# сразу уберем переводы строк и пробелы  
# между тегами из исходного HTML  
>>> html = re.sub(r'>\s+<', '><', html.replace('\n', ''))  
>>> root = BeautifulSoup(html, 'html.parser')  
# найдем теги '<li>' вложенные во второй '<ul>',  
# используя CSS селекторы  
>>> child_ul = root.select('ul > ul > li')  
>>> child_ul  
# [<li>текст 2-1</li>, <li>текст 2-2</li>]  
  
# получаем доступ к родителю  
>>> child_li[0].parent  
# <ul><li>текст 2-1</li><li>текст 2-2</li></ul>  
  
# доступ к родителю родителя  
>>> child_li[0].parent.parent.contents  
[<li>текст 1</li>,<br> <li>текст 2</li>,<br> <ul><li>текст 2-1</li><li>текст 2-2</li></ul>,<br> <li>текст 3</li>]
```

Так же можно перебрать всех родителей элемента с помощью атрибута `.parents`.


```
>>> child_li[0]  
# <li>текст 2-1</li>  
Вверх
```



```
>>> [parent.name for parent in child_li[0].parents]
# ['ul', 'ul', 'div', '[document]']
```

Изменение имен тегов HTML-документа:



 skillfactory.ru
**Курс по пентесту -
от 4 200 р/мес**

от 4 200 ₪ **-40%** **7 000 ₪**

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

Рассрочка >

Преподаватели-практики >

Возврат денег >

Узнать больше

```
# строка нового тега '<a>'
>>> new_tag.string = "Link text"
# добавление тега '<a>' внутрь '<b>'
>>> original_tag.append(new_tag)
>>> original_tag
# <b><a href="http://example.com">Link text.</a></b>
>>> soup
# <p><b><a href="http://example.com">Link text</a></b></p>
```

```
class="boldest">Extremely bold</b></p>', 'html.parser')
```

```
<extremely bold</blockquote>
```

```
>Extremely bold</blockquote></p>
```

```
div>:
```

```
>Extremely bold</b></p>', 'html.parser')
```

```
v>
```

В HTML-документ.

HTML:

```
</b></p>", 'html.parser')
```

```
href="http://example.com")
```

Добавление новых тегов до/после определенного тега или внутрь тега.

```
>>> soup = BeautifulSoup("<p><b>leave</b></p>", 'html.parser')
>>> tag = soup.new_tag("i", id='new')
>>> tag.string = "Don't"
# добавление нового тега <i> до тега <b>
>>> soup.b.insert_before(tag)
>>> soup.b
# <p><i>Don't</i><b>leave</b></p>

# добавление нового тега <i> после тега <b>
>>> soup.b.insert_after(tag)
>>> soup
# <p><b>leave</b><i>Don't</i></p>

# добавление нового тега <i> внутрь тега <b>
>>> soup.b.string.insert_before(tag)
>>> soup.b
# <p><b><i>Don't</i>leave</b></p>
```

Удаление и замена тегов в HTML-документе.


Удаляем тег или строку из дерева HTML:

```
>>> html = '<a href="http://example.com/">I linked to <i>example.com</i></a>'
>>> soup = BeautifulSoup(html, 'html.parser')
>>> a_tag = soup.a
# удаляем HTML-тег '<i>' с сохранением
# в переменной 'i_tag'
>>> i_tag = soup.i.extract()
# Вверх что получилось
>>> a_tag
```

```
# <a href="http://example.com/">I linked to</a>
>>> i_tag
# <i>example.com</i>
```

Заменяем тег и/или строку в дереве HTML:

РЕКЛАМА • 16+



skillfactory.ru

Курс по пентесту -
от 4 200 р/мес

от 4 200 ₽
-40%
7 000 ₽

Получите перспективную профессию в IT. Мини-курс по нейросетям в подарок!

Рассрочка

Преподаватели-практики

Возврат денег

Узнать больше

```
example.com/">I linked to <i>example</i></a>'
html.parser')

# внутри тега `<a>`
)
>>> i_tag
>>> i_tag.replace_with('<b>sample</b></a>')
# <b>sample</b>
```

Тег `<b id = "boldest">` имеет атрибут `id`, значение которого равно `boldest`. Чтобы получить доступ к атрибуту, обращайтесь с тегом как со словарем:

```
id="boldest">bolder</b></p>', 'html.parser')
```

Можно добавлять и изменять атрибуты тега.

```
# изменяем `id`
>>> tag['id'] = 'bold'
# добавляем несколько значений в `class`
>>> tag['class'] = ['new', 'bold']
# или
>>> tag['class'] = 'new bold'
>>> tag
# <b class="new bold" id="bold">bolder</b>
```

А так же производить их удаление.

```
>>> del tag['id']
>>> del tag['class']
>>> tag
# <b>bolder</b>
>>> tag.get('id')
# None
```

Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Основной объект Tag\(\) модуля BeautifulSoup4](#)
- [Методы .find_all\(\) и .find*\(\) модуля BeautifulSoup4](#)
- [CSS селекторы и модуль BeautifulSoup4](#)
- [Метод .get text\(\) модуля BeautifulSoup4](#)
- [Разбор части документа в BeautifulSoup4 Python](#)
- [Ошибки при чтении и разборе HTML модулем BeautifulSoup4](#)