GeekBrains

Методы сбора и обработки данных при помощи Python

Парсинг HTML. Beautiful Soup

На этом уроке

1. Познакомимся с HTML-кодом страниц
2. Изучим структуру DOM.
3. Рассмотрим основы сбора данных с помощью инструмента Beautiful Soup.

Оглавление

HTML

- Теги
- Структура страницы HTML
- Атрибуты тегов и их значения
  - Атрибут class
  - Атрибут id

DOM

- Уровни DOM

Beautiful soup для сбора данных в HTML

- Установка и начало работы
- Дерево синтаксического разбора
- Атрибуты тегов
- Навигация по дереву синтаксического разбора
  - parent
  - contents
  - string
  - next_sibling и previous_sibling
  - next и previous
  - Итерации над объектом Tag
  - Используем имена тегов как элементы
- Поиск в дереве синтаксического разбора
  - Основной метод поиска: find_all()
- Дополнительные методы поиска внутри дерева синтаксического разбора
  - find(name, attrs, recursive, text, **kwargs)

find_next_siblings() и find_next_sibling()

find_previous_siblings() и find_previous_sibling()

find_all_next() и find_next()

find_all_previous() и find_previous()

find_parents() и find_parent()

Практическое задание

1 вариант

2 вариант

Используемая литература

При написании руководства были использованы ОС: Windows 10 x64, Python v3.7.2, PyCharm Edu v2019.3.2

HTML

HTML (HyperText Markup Language) язык гипертекстовой разметки. На нём написано большинство веб-страниц. Язык HTML интерпретируется браузерами, и в результате форматированный текст отображается на экране. HTML-страницы передаются от сервера к клиенту по протоколу HTTP в виде обычного либо зашифрованного текста. Такие страницы обычно имеют расширения html или htm.

Сейчас актуальная версия HTML — 5.2, которая была представлена 14 декабря 2017 года. HTML5 оживил Всемирную паутину, в нём появились новые элементы, например тег <video> для вставки видеопотока и <audio> для аудиопотока. Большинство украшающих тегов, например и <font>, были удалены, так как функции украшения HTML-страницы на себя взяли каскадные таблицы стилей (CSS). HTML5 также предоставляет доступ к API, с которыми можно работать при помощи языка JavaScript.

Теги

HTML-теги — основа языка HTML. Теги используются для разграничения начала и конца элементов в разметке. Каждый HTML-документ состоит из дерева HTML-элементов и текста. Каждый HTML-элемент обозначается начальным (открывающим) <> и конечным (закрывающим)</> тегом. Открывающий и закрывающий теги содержат имя тега.
Все HTML-элементы делятся на пять типов:

- **пустые элементы** — <area>, <base>,
, <col>, <embed>, <hr>, <img>, <input>, <link>, <menuitem>, <meta>, <param>, <source>, <track>, <wbr>;
- **элементы с неформатированным текстом** — <script>, <style>;
- **элементы, выводящие неформатированный текст** — <textarea>, <title>;
- **элементы из другого пространства имён** — MathML и SVG;
- **обычные элементы** — все остальные элементы.

Важно!

Внутри тегов хранится большинство необходимой информации, поэтому важно не только находить данные правильно, но и верно определять, в каких тегах они находятся.

Структура страницы HTML

Пример HTML-страницы:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>
            (Это title) Пример страницы на HTML5
        </title>
    </head>
    <body>
        <header>
            <hgroup>
                <h1>
                    Заголовок "h1" из hgroup
                </h1>
                <h2>
                    Заголовок "h2" из hgroup
                </h2>
            </hgroup>
        </header>
        <nav>
            <menu>
                <li>
                    <a href="link1.html">
                        Первая ссылка из блока "nav"
                    </a>
                </li>
                <li>
                    <a href="link2.html">
                        Вторая ссылка из блока "nav"
                    </a>
                </li>r
            </menu>
        </nav>
        <section>
            <article>
                <h1>
                    Заголовок статьи из блока "article"
                </h1>
                <p>
                    Текст абзаца статьи из блока "article"
                </p>
                <details>
```

```
                    <summary>
                        Блок "details", текст тега "summary"
                    </summary>
                    <p>
                        Абзац из блока "details"
                    </p>
                </details>
            </article>
        </section>
        <footer>
            <time>
                Содержимое тега "time" блока "footer"
            </time>
            <p>
                Содержимое абзаца из блока "footer"
            </p>
        </footer>
    </body>
</html>
```

Разберём эту страницу построчно:

- <!DOCTYPE html> — здесь мы объявляем, что это документ HTML версии 5. Для 4-й версии HTML эта строка будет следующей:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN""http://www.w3.org/TR/html4/strict.dtd">
```

- html — элемент, включающий всю веб-страницу;
- head — головная часть HTML-страницы: в ней обычно описываются метатеги документа, подключаются дополнительные таблицы стилей и JavaScript, объявляется title страницы (текст, который будет выведен в заголовке окна или вкладки браузера) — всё, что касается мета- и описательной информации HTML-страницы;
- body — тег, внутри которого описывается основное содержимое страницы;
- header — именованный тег, определяющий заглавие страницы. Содержит в себе теги hgroup — логическое объединение заголовков, если их несколько;
- hX — заголовок, где X — число от 1 до 6 (по стандарту). X — индекс, обозначающий уровень заголовка: чем меньше число, тем главнее заголовок и тем крупнее шрифт по умолчанию;
- nav — именованный навигационный тег, внутри которого обычно располагают элементы для навигации, такие как menu, ul, li;
- section — элемент, крайне распространённый в одностраничных приложениях и посадочных страницах.

Полный список тегов с удобной группировкой — в руководстве: HTML 5.2. теги: HTML5BOOK.RU

Атрибуты тегов и их значения

У тега могут быть свойства, называемые *атрибутами,* дающие дополнительные возможности форматирования текста. Они записываются в виде сочетания имени атрибута и значения, причём текстовые значения заключаются в кавычки.

Например, можно выделить фрагмент текста определённым шрифтом, используя тег <font> и указав в этом теге название шрифта **face** и желаемый размер **size**:

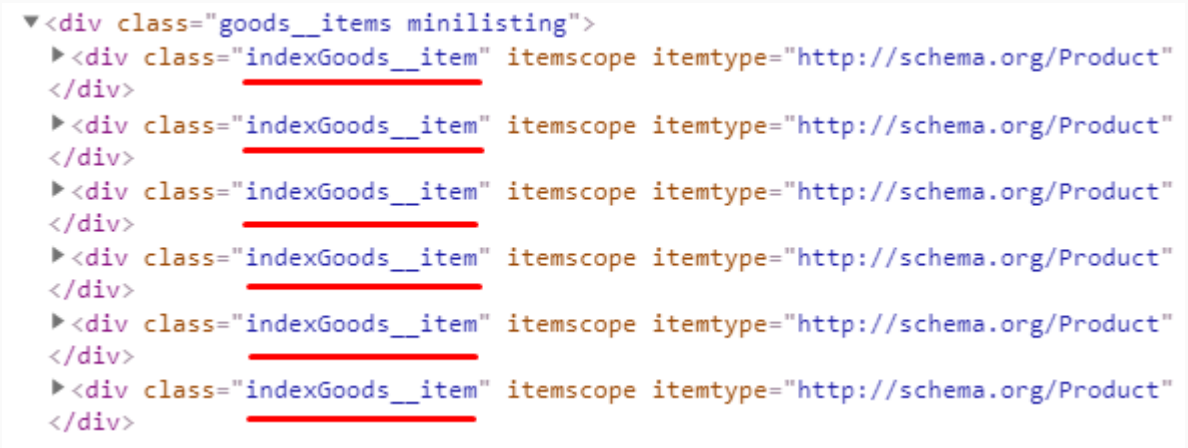
```
<font face="Times, Arial, Courier" size=4> Оформляемый текст </font>
```

Важно!

Обращайте внимание на атрибуты: они помогают идентифицировать нужные вам теги и очень часто делают их уникальными, тем самым упрощая методы их поиска.

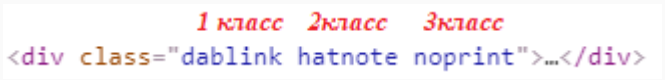
Атрибут class

С помощью классов очень часто задают стили для одного и более элементов, объединенных каким то логическим смыслом (например товары из категории «продукты» в интернет магазине). Для этого чаще всего стараются использовать одинаковые имена класса в разных местах страницы или даже на разных страницах сайта:



Важно!

*Один элемент может имеет **несколько** атрибутов класса! В этом случае их значения перечисляются через пробел.*



Атрибут id

Идентификатор однозначно определяет **один** конкретный элемент. Значение id должно быть уникальным, на одной странице может встречаться только один раз. Благодаря этому атрибуту мы можем однозначно точно выбрать именно нужный нам элемент на странице.

```
<div id="mw-page-base" class="noprint"></div>
<div id="mw-head-base" class="noprint"></div>
```

На рисунке классы у элементов одинаковые, но id — разные.

DOM

DOM (от англ. *Document Object Model* — «объектная модель документа») — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого может быть:

- элементом,
- атрибутом,
- текстовым, графическим или любым другим объектом.

Узлы связаны между собой отношениями «родительский-дочерний».

Изначально различные браузеры имели собственные модели документов (DOM), несовместимые с остальными. Для обеспечения взаимной и обратной совместимости специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.

Уровни DOM

Модель DOM имеет несколько уровней. Каждый новый уровень расширяет функционал предыдущего:

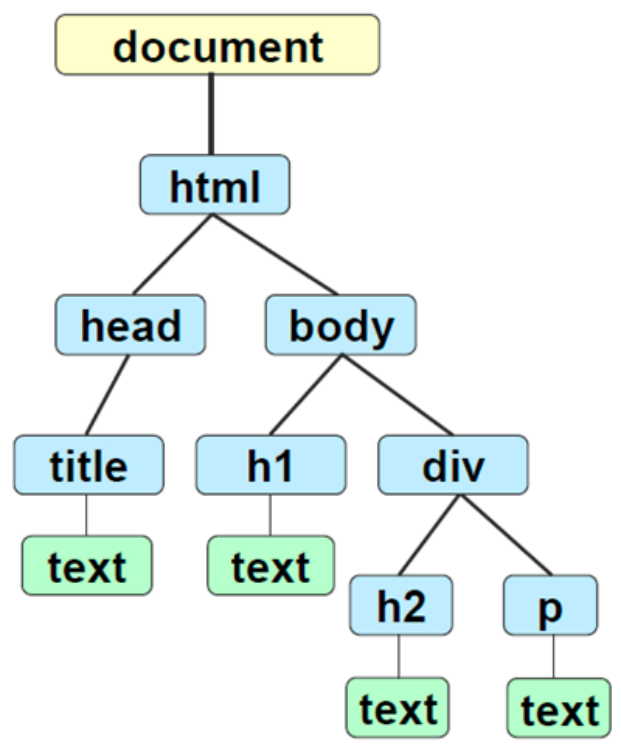
1. Уровень 1 включает в себя поддержку XML 1.0 и HTML (управление деревом, перемещение по дереву) без пространства имён (этот уровень существовал ещё до разделения модели на уровни).
2. Уровень 2 поддерживает пространства имён XML и CSS.
3. Уровень 3 состоит из 6 спецификаций:
    - a. DOM Level 3 Core.
    - b. DOM Level 3 Load and Save.
    - c. DOM Level 3 XPath.
    - d. DOM Level 3 Views and Formatting.
    - e. DOM Level 3 Requirements.
    - f. DOM Level 3 Validation.

Для лучшего понимания DOM рассмотрим пример простой HTML-страницы:

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>PAGE</title>
    </head>
    <body>
        <h1>
            PAGE
        </h1>
        <div>
            <h2>Раздел</h2>
            <p>Параграф</p>
        </div>
```

```
    </body>
</html>
```

В виде DOM она будет представлена так:



В представленном дереве два типа узлов:

- теги — узлы дерева, за их счёт выстроена структура дерева, его вложенность;
- текст — обозначены как text, содержат исключительно строку текста.

При построении модели DOM учитываются и переносы строк, и пробелы.

Beautiful Soup для сбора данных в HTML

Установка и начало работы

Парсить HTML мы будем с применением парсера lxml, который преобразует наш HTML-код в DOM, а обрабатывать полученную структуру — с помощью библиотеки Beautiful Soup. Для начала работы установим их, введя команды в терминале:

```
pip install bs4

pip install lxml
```

Подключить Beautiful Soup к вашему приложению можно с помощью одной из ниже приведенных строк:

```
from bs4 import BeautifulSoup          #Для обработки HTML
from bs4 import BeautifulStoneSoup     #Для обработки XML
import bs4                             #Для обработки и того и другого
```

Для просмотра основных возможностей Beautiful Soup будем рассматривать простую HTML-страницу:

```
<html>
    <head>
        <title>
            Page title
        </title>
    </head>
    <body>
        <p id="firstpara" align="center">
            This is paragraph <b>one</b>.
        </p>
        <p id="secondpara" align="blah">
            This is paragraph <b>two</b>.
        </p>
    </body>
</html>
```

Чтобы начать работу с Beautifull Soup, необходимо выполнить get-запрос, и распарсим текст ответа от сервера с помощью подключенного парсера lxml:

```
from bs4 import BeautifulSoup as bs
import requests
response = requests.get('http://example.com').text
                          #получаем экземпляр класса bs
```

Дерево синтаксического разбора

Дерево синтаксического разбора — структуры данных Beautiful Soup, которые создаются по мере синтаксического разбора документа. Объект парсера (экземпляр класса BeautifulSoup или BeautifulStoneSoup) обладает большой глубиной вложенности связанных структур данных, соответствующих структуре документа XML или HTML. Объект парсера состоит из объектов двух других типов:

- объектов **Tag**, которые соответствуют тегам, к примеру, тегу и тегу <p>;
- объекты **NavigableString**, соответствующие таким строкам, как Page title или This is paragraph.

Атрибуты тегов

Теги HTML имеют атрибуты: например, каждый тег <p> в приведенном выше примере HTML имеет атрибуты id и align. К атрибутам тегов можно обращаться таким же образом, как если бы объект Tag был словарём:

```
firstPTag, secondPTag = soup.find_all('p')
firstPTag['id']  # 'firstPara'
secondPTag['id'] # 'secondPara'
```

Атрибуты есть только у объектов **Tag**. Объекты **NavigableString** не имеют атрибутов.

Навигация по дереву синтаксического разбора

Важно!

*Все объекты **Tag** содержат элементы, перечисленные ниже (тем не менее, **фактическое значение элемента может равняться None**). Объекты NavigableString имеют все из них, за исключением contents и string.*

parent

В примере выше родителем объекта <HEAD> Tag будет объект <HTML> Tag. Родитель объекта <HTML> Tag — сам объект парсера BeautifulSoup. Родитель объекта парсера равен None. Передвигаясь по объектам parent, можно перемещаться по дереву синтаксического разбора:

```
soup.head.parent.name
# 'html'
soup.head.parent.parent.__class__.__name__
# 'BeautifulSoup'
soup.parent == None
# True
```

contents

Parent перемещает нас вверх по дереву синтаксического разбора. С помощью **contents** мы перемещаемся вниз по дереву синтаксического разбора. **Contents — упорядоченный список**, состоящийTag — из объектов **Tag** и **NavigableString**, содержащихся в элементе страницы (**page element**).

Только объект парсера самого высокого уровня и объекты **Tag** содержат **contents**. Объекты **NavigableString** — простые строки и не могут содержать подэлементов, поэтому они не содержат **contents**.

В примере выше элемент contents первого объекта <P> это список, содержащий:

- объект NavigableString (This is paragraph);
- объект <B> Tag;
- и ещё один объект NavigableString (.).

Элемент **contents** объекта <B> Tag: список, состоящий из од ного объекта **NavigableString** (one).

```
pTag = soup.p
pTag.contents
# ['This is paragraph ', one, '.']
pTag.contents[1].contents
# ['one']
pTag.contents[0].contents
# AttributeError: 'NavigableString' object has no attribute 'contents'
```

string

Если у тега есть только один дочерний узел, который будет строкой, этот узел будет доступен через tag.string точно так же, как и через tag.contents[0]. В примере выше soup.b.string — объект **NavigableString**, отображающий Unicode-строку one. Эта строка содержится в первом объекте <B> Tag дерева синтаксического разбора.

```
soup.b.string
# 'one'
soup.b.contents[0]
# 'one'
```

Но soup.p.string равен None, поскольку первый объект <P> Tag в дереве синтаксического разбора имеет более одного дочернего элемента. soup.head.string также равен None, хотя объект <HEAD> Tag имеет только один дочерний элемент, поскольку этот элемент — Tag (объект <TITLE> Tag), а не **NavigableString**.

```
soup.p.string == None
# True
soup.head.string == None
# True
```

next_sibling и previous_sibling

Эти элементы позволяют пропускать следующий или предыдущий элемент на этом же уровне дерева синтаксического разбора. В представленном выше документе элемент **next_sibling** объекта <HEAD> Tag равен объекту <BODY> Tag, так как <BODY> Tag — следующий вложенный элемент по отношению к объекту <html> Tag. Элемент **next_sibling** объекта <BODY> Tag равен None, поскольку в нём больше нет вложенных по отношению к объекту <HTML> Tag элементов.

```
soup.head.next_sibling.name
# 'body'
```

```
soup.html.next_sibling == None
# True
```

И наоборот, элемент previousSibling объекта <BODY> Tag равен объекту <HEAD> tag, а элемент previousSibling объекта <HEAD> Tag равен None:

```
soup.body.previous_sibling.name
# 'head'
soup.head.previous_sibling == None
# True
```

Несколько примеров: элемент **next_sibling** первого объекта <P> Tag равен второму объекту <P> Tag. Элемент **previous_sibling** объекта <B> Tag внутри второго объекта <P> Tag равен объекту **NavigableString** This is paragraph. Элемент **previous_sibling** объекта **NavigableString** равен None внутри первого объекта <P> Tag.

```
soup.p.next_sibling
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>

secondBTag = soup.find_all('b')[1]
secondBTag.previous_sibling
# 'This is paragraph'

secondBTag.previous_sibling.previous_sibling == None
# True
```

next и previous

Эти элементы позволяют передвигаться по элементам документа в том порядке, в котором они были обработаны парсером, а не в порядке появления в дереве. Например, элемент next для объекта <HEAD> Tag равен объекту <TITLE> Tag, а не объекту <BODY> Tag. Это потому, что в исходном документе, тег <TITLE> идет сразу после тега <HEAD>.

```
soup.head.next
# 'title'
soup.head.next_sibling.name
# 'body'
soup.head.previous.name
# 'html'
```

Поскольку элементы **next** и **previous** связаны, содержимое элемента **contents** объекта Tag обновляется до того, как элемент **next_sibling**. Как правило, эти элементы не используют, но иногда это наиболее быстрый способ получить что-либо скрытое в глубине дерева синтаксического разбора.

Итерации над объектом Tag

Над элементом **contents** объекта Tag можно производить итерации, рассматривая его в качестве списка. Это полезное упрощение. Подобным образом можно узнать, сколько дочерних узлов имеет объект Tag, вызвав функцию len(tag) вместо len(tag.contents). В терминах документа выше:

```
for i in soup.body:
    print (i)
    # <p id="firstpara" align="center">This is paragraph <b>one</b>.</p>
    # <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>
len(soup.body)
# 2
len(soup.body.contents)
# 2
```

Используем имена тегов как элементы

Гораздо легче перемещаться по дереву синтаксического разбора, если в качестве имён тегов выступали элементы парсера или объекта Tag. Будем так делать в новых примерах. В терминах документа выше, soup.head возвращает нам первый (как и следовало ожидать, единственный) объекта <HEAD> Tag документа:

```
soup.head
# <head><title>Page title</title></head>
```

Вызов mytag.foo возвратит первого потомка mytag, чем, как ни странно, будет объект <FOO> Tag. Если каких-либо объектов <FOO> Tag внутри mytag нет, то mytag.foo возвратит None. Вы можете использовать это для очень быстрого обхода дерева синтаксического разбора:

```
soup.head.title
# <title>Page title</title>
soup.body.p.b.string
# 'one'
```

Также это можно использовать для быстрого перехода к определенной части дерева синтаксического разбора. Например, если вас не беспокоит отсутствие тегов <TITLE> на предусмотренных для них местах, вы можете просто использовать soup.title для получения названия документа HTML. Вам не нужно использовать soup.head.title:

```
soup.title.string
# 'Page title'/
```

- soup.p перейдёт к первому тегу <P> внутри документа, где бы тот ни был;
- soup.table.tr.td перейдёт к первому столбцу первой строки первой же таблицы документа.

Фактически эти элементы — алиасы для метода first, описываемого ниже. Мы упоминаем их здесь потому, что алиасы делают очень лёгким увеличение масштаба интересующей вас части хорошо знакомого дерева синтаксического разбора.

Альтернативная форма этого стиля позволяет обращаться к первому тегу <FOO> .fooTag вместо .foo. Например, soup.table.tr.td можно также отобразить как soup.tableTag.trTag.tdTag или даже soup.tableTag.tr.tdTag.

Поиск в дереве синтаксического разбора

Beautiful Soup предоставляет множество методов для обхода дерева синтаксического разбора, отбирая по заданным критериям объекты Tag и NavigableString. Для определения критериев отбора объектов Beautiful Soup есть несколько способов. Продемонстрируем доскональное исследование наиболее общего из всех методов поиска Beautiful Soup, find_all. Как и раньше, показывать будем на следующем документе:

```
<html>
    <head>
        <title>
            Page title
        </title>
    </head>
    <body>
        <p id="firstpara" align="center">
            This is paragraph <b>one</b>.
        </p>
        <p id="secondpara" align="blah">
            This is paragraph <b>two</b>.
        </p>
    </body>
</html>
```

Важно!

Оба описываемых в этом разделе метода (find_all и find) доступны только для объектов Tag и объектов парсера самого высокого уровня, но не для объектов NavigableString.

Основной метод поиска: find_all()

Метод обхода дерева find_all начинает с заданной точки и ищет все объекты Tag и NavigableString, соответствующие заданным критериям. Сигнатура метода find_all следующая:

```
find_all(name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)
```

Эти аргументы появляются снова и снова повсюду в Beautiful Soup API. Наиболее важные — аргументы name и именованные аргументы.

Аргумент name ограничивает набор имен тегов. Есть несколько способов ограничить имена и все они появляются снова и снова повсюду в Beautiful Soup API:

1. Самый простой способ — передать имя тега. Приведём код, который ищет все объекты <B> Tag в документе:

```
soup.find_all('b')
# [<b>one</b>, <b>two</b>]
```

2. Также можно передать регулярное выражение. Код, который ищет все теги, имена которых начинаются на английскую букву B:

```
import re
tagsStartingWithB = soup.find_all(re.compile('^b'))
tag.name for tag in tagsStartingWithB
# ['body', 'b', 'b']
```

3. Можно передать список или словарь. Эти два вызова ищут все теги <TITLE> и <P>. Принцип работы у них одинаков, но второй вызов отработает быстрее:

```
soup.find_all(['title', 'p'])
# [<title>Page title</title>,
# <p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]

soup.find_all({'title' : True, 'p' : True})
# [<title>Page title</title>,
# <p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]
```

4. Можно передать специальное значение True, которому соответствуют все теги с любыми именами.

```
allTags = soup.find_all(True)
tag.name for tag in allTags
# ['html', 'head', 'title', 'body', 'p', 'b', 'p', 'b']
```

Это не выглядит полезным, но True необходим, когда нужно ограничить значения атрибутов.Можно передать вызываемый объект, который принимает объект Tag как единственный аргумент и возвращает логическое значение. Каждый объект Tag, который находит find_all, будет передан в этот объект, и если его вызов возвращает True, то необходимый тег найден.

Вот код, который ищет теги с двумя и только двумя атрибутами:

```
soup.find_all(lambda tag: len(tag.attrs) == 2)
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]
```

Данный код ищет теги, имена которых состоят из одной букву и которые не имеют атрибутов:

```
soup.find_all(lambda tag: len(tag.name) == 1 and not tag.attrs)
# [<b>one</b>, <b>two</b>]
```

Аргументы ключевых слов (keyword arguments) налагают ограничения на атрибуты тега. Вот простой пример поиска всех тегов, атрибут которых align имеет значение center:

```
soup.find_all(align="center")
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>]
```

Как и в случае с аргументом name, вы можете передать именованный аргумент различными видами объектов для наложения разных ограничений на соответствующие атрибуты. Можно передать строку, как показано выше, чтобы ограничить значение атрибута единственным значением. Можно также передать регулярное выражение, список, хеш, специальные значения True или None, или вызываемый объект, который получает значение атрибута в качестве аргумента (обратите внимание на то, что значение может быть и None). Несколько примеров:

```
soup.find_all(id=re.compile("para$"))
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]

soup.find_all(align=["center", "blah"])
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]
soup.find_all(align=lambda(value): value and len(value) < 5)
# [<p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]
```

Специальные значения True и None особо интересны. Значению True соответствует тег, заданный атрибут которого имеет любое значение, а None соответствует тег, у которого заданный атрибут не содержит значения. Несколько примеров:

```
soup.find_all(align=True)
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]

[tag.name for tag in soup.find_all(align=None)]
# ['html', 'head', 'title', 'body', 'b', 'b']
```

Если необходимо наложить сложные или взаимосвязанные ограничения на атрибуты тегов, передавайте вызываемый объект для name, как показано выше, и работайте с объектом Tag.

Здесь вы должны обратить внимание на одну проблему. Что делать, если в вашем документе есть тег, который определяет атрибут с именем name? Поскольку методы поиска Beautiful Soup всегда определяют аргумент name, вы не можете использовать именованный аргумент с именем name. В качестве именованного аргумента также нельзя использовать зарезервированные слова Python, такие как for.

Beautiful Soup предоставляет специальный аргумент с именем attrs, который можно использовать в таких ситуациях. attrs представляет собой словарь, который работает также как именованные аргументы:

```
soup.find_all(id=re.compile("para$"))
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]

soup.find_all(attrs={'id' : re.compile("para$")})
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]
```

Можно использовать attrs, если необходимо наложить ограничения на атрибуты, имена которых совпадают с зарезервированными словами Python — class, for или import; или атрибуты, имена которых — неименованные аргументы методов поиска Beautiful Soup: name, recursive, limit, text или сам attrs.

Установка аргумента **limit** позволяет останавливать поиск после того, как будет найдено заданное число совпадений. Если в документе тысячи таблиц, а нужно только четыре, то передайте в аргументе limit значение 4 и сэкономьте время. По умолчанию ограничения нет.

```
soup.find_all('p', limit=1)
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>]

soup.find_all('p', limit=100)
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>,
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>]
```

Дополнительные методы поиска внутри дерева синтаксического разбора

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

Метод find почти в точности совпадает с findAll, за исключением того, что он ищет первое вхождение искомого объекта, а не все. Это похоже на установку для результирующего множества limit равным 1 и затем извлечения единственного результата из массива.

Сигнатура метода find:

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

Но метод .find() не получает limit и возвращает единственный результат:

```
soup.findAll('p', limit=1)
# [<p id="firstpara" align="center">This is paragraph <b>one</b>.</p>]

soup.find('p', limit=1)
# <p id="firstpara" align="center">This is paragraph <b>one</b>.</p>

soup.find('nosuchtag', limit=1) == None
# True
```

find_next_siblings() и find_next_sibling()

Эти методы часто следуют за элементом next_sibling, собирая объекты Tag или NavigableText, соответствующие заданным критериям. Сигнатуры методов:

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

Примеры вызова методов:

```
paraText = soup.find(text='This is paragraph ')
paraText.find_next_siblings('b')
# [<b>one</b>]

paraText.find_next_sibling(text = lambda(text): len(text) == 1)
# '.'
```

find_previous_siblings() и find_previous_sibling()

Эти методы часто следуют за элементом previous_sibling, собирая объекты Tag или NavigableText, соответствующие заданным критериям. Сигнатуры методов:

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

Примеры вызова методов:

```
paraText = soup.find(text='.')
paraText.find_previous_siblings('b')
# [<b>one</b>]
paraText.find_previous_sibling(text = True)
# 'This is paragraph '
```

find_all_next() и find_next()

Эти методы часто следуют за элементом Next, собирая объекты Tag или NavigableText, соответствующие заданным критериям. Сигнатуры методов:

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

Примеры:

```
pTag = soup.find('p')
pTag.find_all_next(text=True)
# ['This is paragraph ', 'one', '.', 'This is paragraph ', 'two', '.']

pTag.find_next('p')
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>

pTag.find_next('b')
# <b>one</b>
```

find_all_previous() и find_previous()

Эти методы часто следуют за элементом previous, собирая объекты Tag или NavigableText, соответствующие заданным критериям. Сигнатуры методов:

```
find_all_previous(name, attrs, text, limit, **kwargs)
find_previous(name, attrs, text, **kwargs)
```

Примеры:

```
lastPTag = soup('p')[-1]
lastPTag.find_all_previous(text=True)
# ['.', 'one', 'This is paragraph ', 'Page title']
# Note the reverse order!

lastPTag.find_previous('p')
# <p id="firstpara" align="center">This is paragraph <b>one</b>.</p>

lastPTag.find_previous('b')
# <b>one</b>
```

find_parents() и find_parent()

Эти методы часто следуют за элементом parent, собирая объекты Tag или NavigableText, соответствующие заданным критериям. Они не принимают аргумент text, поскольку не может быть объектов, предком которых был бы объект NavigableString. Сигнатуры методов:

```
bTag = soup.find('b')
[tag.name for tag in bTag.find_parents()]
# ['p', 'body', 'html', '[document]']
# NOTE: "'[document]'" means that that the parser object itself matched.
bTag.find_parent('body').name
# 'body'
```

Примеры:

```
bTag = soup.find('b')
```

```
[tag.name for tag in bTag.find_parents()]
# ['p', 'body', 'html', '[document]']
# NOTE: "'[document]'" means that that the parser object itself matched.
bTag.find_parent('body').name
# 'body'
```

Глоссарий

HTML (HyperText Markup Language) — HTML (HyperText Markup Language) язык гипертекстовой разметки. Он интерпретируется браузерами, и в результате форматированный текст отображается на экране. HTML-страницы передаются от сервера к клиенту по протоколу HTTP в виде обычного либо зашифрованного текста.

HTML-теги — используются для разграничения начала и конца элементов в разметке.

Атрибуты — свойства тега, дающие дополнительные возможности форматирования текста.

DOM (Document Object Model) — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Библиотека Beautiful Soup — парсер lxml, который преобразует наш HTML-код в DOM и обрабатывает полученную структуру.

Дерево синтаксического разбора — структуры данных Beautiful Soup, которые создаются по мере синтаксического разбора документа.

Дополнительные материалы

1. Полезные примеры использования методов Soup.
2. Парсим данные каталога сайта, используя Beautiful Soup (1).
3. Парсим данные катачастьлога сайта, используя Beautiful Soup и Selenium (часть 2).
4. Парсим Википедию с Beautiful Soup 4.

Домашнее задание

Необходимо собрать информацию о вакансиях на вводимую должность (используем input или через аргументы) с сайтов Superjob и HH. Приложение должно анализировать несколько страниц сайта (также вводим через input или аргументы). Получившийся список должен содержать в себе минимум:

- Наименование вакансии.
- Предлагаемую зарплату (отдельно минимальную и максимальную).
- Ссылку на саму вакансию.

- Сайт, откуда собрана вакансия.

По желанию можно добавить ещё параметры вакансии (например, работодателя и расположение). Структура должна быть одинаковая для вакансий с обоих сайтов. Общий результат можно вывести с помощью dataFrame через pandas.

Можно выполнить по желанию один любой вариант или оба при желании и возможности.

Используемая литература

1. Базовая структура HTML.
2. Элементы HTML/CSS.
3. Объектная модель документа.
4. Стандарт HTML5.
5. Документация Beautifull Soup.
6. Документации/BeautifulSoup.