Парсинг web-страниц на Python

Для того, чтобы выполнять анализ данных, необходимо располагать этими самыми данными. Пожалуй, самый большой источник актуальных данных, доступный на сегодня – это интернет. С помощью языка Python очень удобно извлекать данные из web-страниц. Это может быть текст статей, афиша кинотеатров, цены на мобильные телефоны, различные отзывы и комментарии, данные о погоде в регионе и т.п.

При выполнении практики вы научитесь извлекать данные с web-страницы и выполнять их простейшую обработку.

Структура web-страницы и Document Object Model

Web-страницы представляют собой документы HTML. HTML – теговый язык разметки документов. Любой документ на языке HTML представляет собой набор элементов, причём начало и конец каждого элемента обозначается специальными пометками – тегами. Элементы могут быть пустыми, то есть не содержащими никакого текста и других данных (например, тег перевода строки **
). В этом случае обычно не указывается закрывающий тег. Кроме того, элементы могут иметь атрибуты, определяющие какие-либо их свойства (например, размер шрифта для элемента **font). Атрибуты указываются в открывающем теге. Вот примеры фрагментов HTML-документа:

```
<strong>Текст между двумя тегами – открывающим и закрывающим.</strong>
<a href="http://www.example.com">Элемент содержит href – гиперссылку.</a>
```

А вот пример пустого элемента:

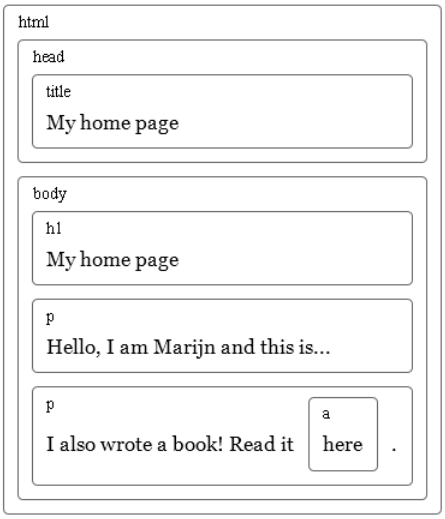
```
<br>
```

Регистр, в котором набрано имя элемента и имена атрибутов, в HTML значения не имеет (в отличие от XHTML). Элементы могут быть вложенными. Теги вроде **<body>** и **</body>** включают в себя другие теги, которые в свою очередь включают теги, или текст.

Рассмотрим пример документа:

```
<!doctype html>
<html>
  <head>
    <title>Моя домашняя страничка</title>
  </head>
  <body>
    <h1> Моя домашняя страничка </h1>
    <p>Привет, я Марийн и это моя домашняя страничка.</p>
    <p>А ещё я книжку написал! Читайте её
      <a href="http://eloquentjavascript.net">здесь</a>.</p>
  </body>
</html>
```

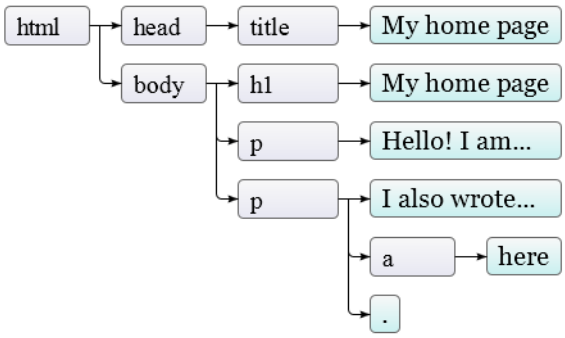
У этой страницы следующая структура:



Структура данных, использующаяся браузером для представления документа, отражает его форму. Для каждого элемента есть объект, с которым мы можем взаимодействовать и узнавать про него разные данные – какой тег он представляет, какие элементы и текст содержит. Это представление называется **`Document Object Model`** (объектная модель документа), или сокращённо **`DOM`**. Мы можем получить доступ к этим объектам через глобальную переменную **`document`**.

Документ DOM можно представить в виде дерева. Узлы для обычных элементов, представляющих теги HTML, определяют структуру документа. У них могут быть дочерние узлы. Пример такого узла – **`document.body`**. Некоторые из этих дочерних узлов могут оказаться листьями – например, текст или комментарии.

Ниже представлен способ представить дерево документа графически:



Каскадные таблицы стилей

CSS (англ. Cascading Style Sheets – каскадные таблицы стилей) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана), или при выводе устройствами, использующими шрифт Брайля.

С помощью CSS-селекторов можно выбирать любые элементы web-страницы. Ниже приведены примеры основных селекторов:

```
*
```

Символ звездочки позволяет выбрать все элементы на странице.

```
#X
```

Символ решетки позволяет нам отбирать элементы по идентификатору. Это один из наиболее распространенных способов отбора элементов.

```
.X
```

Это селектор класса. Разница между id и классами в том, что с помощью классов можно выбирать сразу несколько элементов. Используется, если нужно применить один стиль к группе элементов.

```
X Y
```

Следующий часто используемый тип селектора – селектор потомка. Его следует использовать, когда нужно производить более точечный отбор элементов. Такой селектор можно применить, например, если нужно выбрать не все тэги ссылок, а только те, что находятся внутри неупорядоченного списка.

```
X[href="foo"]
```

Код выше позволит найти все элементы, атрибут **`href`** у которых равен **`foo`**.

Для экспериментов с CSS-селекторами прямо в браузере стоит запустить инструменты разработчика, затем консоль. И в консоли использовать команду **`$$("CSS-селектор")`**.

Дополнительную информацию по использованию CSS-селекторов можно найти в интернете. Например, здесь: http://www.codeharmony.ru/materials/42.

Кроме CSS-селекторов получить доступ к элементам можно с помощью XPath, но для выполнения практики и в большинстве реальных задач достаточно использовать CSS.

Установка модуля Beautiful Soup 4

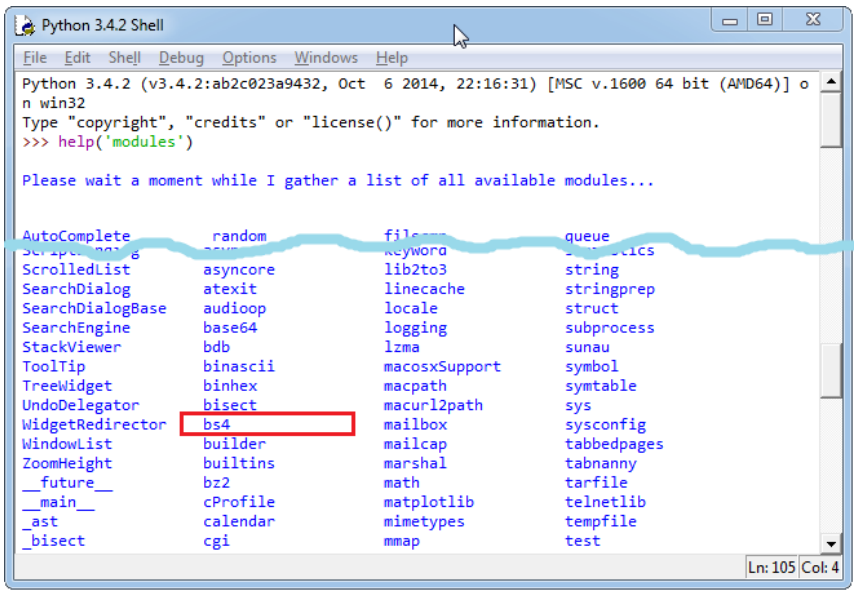
Для парсинга web-страниц мы будем использовать модуль **`Beautiful Soup 4`**. Проще всего установить его через **`cmd`** с помощью команды:

```
pip install beautifulsoup4
```

После завершения установки необходимо запустить интерпретатор Python и выполнить команду

```
help('modules').
```

Если установка прошла успешно, в списке установленных модулей вы увидите модуль **`bs4`** (см. ниже).



Пример: разбор статьи с habrahabr.ru

Для начала нужно получить html-страницу и разобрать содержимое с помощью **Beautiful Soup**. Это можно сделать с помощью кода, приведенного ниже. В данном примере мы загружаем страницу из файла на диске, но можно получить страницу и из интернет. Статья, с которой мы работаем в этой практике, получена с адреса https://habrahabr.ru/post/252379/.

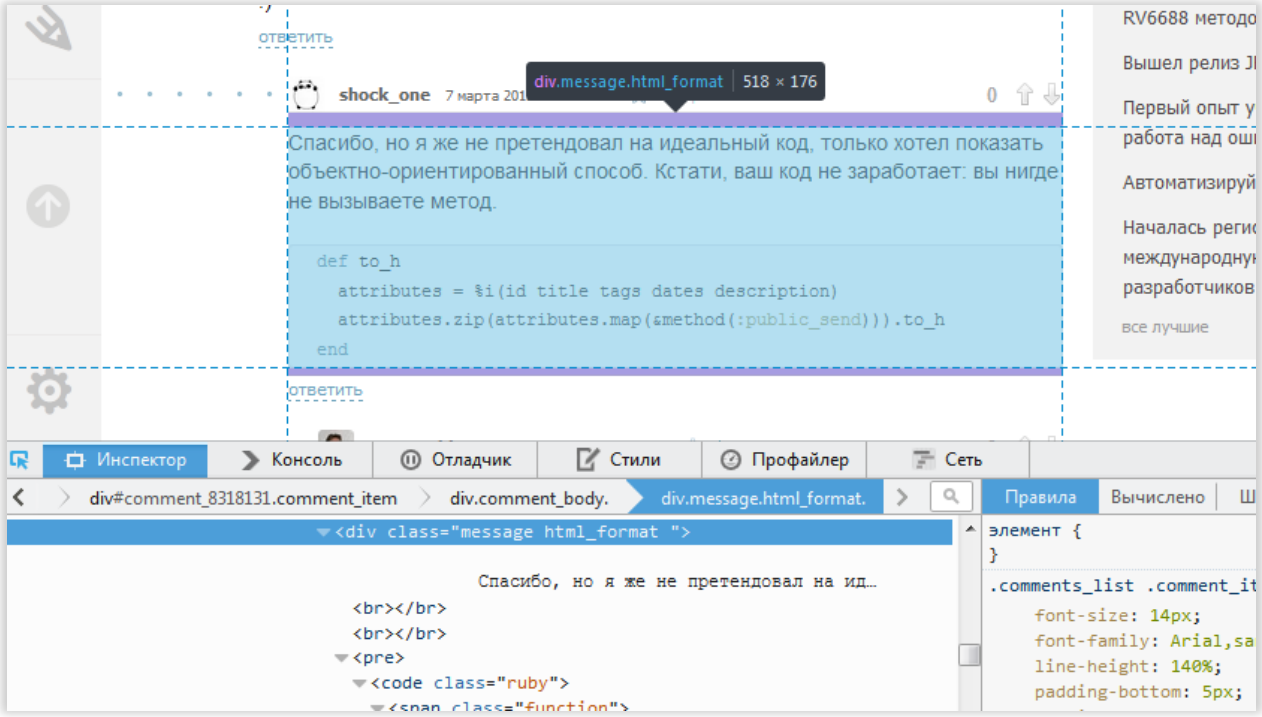
```
from bs4 import BeautifulSoup as bs
import codecs

# открываем документ
doc = bs(codecs.open('article.htm', encoding='utf-8', mode='r').read(), 'html.parser')
```

В первой строке импортируется модуль **Beautiful Soup**, которому даётся псевдоним **bs**, чтобы не засорять пространство имён.

Во второй строке импортируется модуль **codecs**, который необходим для корректной работы с кодировкой html-страницы.

Далее в последней строке выполняется открытие html-файла (с кодировкой **utf-8**) и парсинг его с помощью **Beautiful Soup**. Результат сохраняется в переменной **doc**, в которой хранится дерево **DOM**, к которому далее мы будем применять CSS-селекторы.

После того, как документ открыт и обработан, можно извлекать из него полезную информацию. Но для этого необходимо знать структуру документа. Узнать ее можно с помощью инструментов разработки, встроенных в современные браузеры. В **Google Chrome** и **Mozilla Firefox** инструменты вызываются по **F12** и выглядят, как показано ниже (для Firefox):



Здесь можно изучить структуру документа, найти в структуре интересующий элемент, просмотреть информацию о применяемых стилях, поменять значения атрибутов, отлаживать JavaScript-код и т.д.

Для того, чтобы поэкспериментировать с CSS-селекторами и **XPath** прямо в браузере, необходимо в инструментах разработчика отрыть консоль, и в консоли выполнять команды:

```
$$("CSS-селектор")                  <- для применения CSS-селекторов
$x("выражение XPath")               <- для применения XPath
```

Далее извлечем для примера информацию о статье: автор, заголовок, дата публикации, список тегов, рейтинг.

```
# извлечение данных из статьи
author = doc.select('.author-info__name')[0].decode_contents().strip()
title = doc.select('.post__title span')[1].decode_contents().strip()
date = doc.select('.post__time_published')[0].decode_contents().strip()
tags = list(map(lambda x: x.decode_contents().strip(), doc.select('div.post__tags ul li a')))
rating = int(doc.select('div.post-additionals span.voting-wjt__counter-score.js-
score')[0].decode_contents().strip())

# вывод на экран
print('Автор:', author)
print('Заголовок:', title)
print('Дата:', date)
print('Теги:', tags)
print('Рейтинг:', rating)
```

Метод **select** документа применяет CSS-селекторы. С помощью них можно находить элементы дерева **DOM** с указанными тегами (включая вложенность тегов), классами, идентификаторами. Например, строка **doc.select('.post__time_published')** найдет все элементы с классом **post__time_published**, а строка

doc.select('div.post-additionals span.voting-wjt__counter-score.js-score')

найдет все элементы **span** с классами **voting-wjt__counter-score** и **js-score**, расположенные внутри тега **div** с классом **post-additionals**.

Метод **decode_contents** позволяет получить текст элемента.

Метод **strip** удаляет непечатаемые символы из начала и конца строки.

После вызова **doc.select** добавляется целочисленный индекс. Это делается потому, что метод **select** возвращает список узлов, а нам необходим конкретный узел из этого списка. В третьей строке берётся второй узел списка (с индексом 1), так как это обусловлено структурой html-страницы, с которой мы работаем. Структура была исследована с помощью инструментов разработчика в браузере.

Теперь попробуем получить информацию о комментариях. Допустим, нас интересует количество комментариев, самый короткий комментарий, самый популярный комментарий и самый активный комментатор. Скрипт для получения этих данных приведен ниже:

```
# извлечение данных о комментариях
comments = []
for node in doc.select('div.comment_body'):
    text = node.select('div.message')[0].decode_contents().strip()
    rating = int(node.select('span.voting-wjt__counter-score.js-
score')[0].decode_contents().strip())
    author = node.select('a.comment-item__username')[0].decode_contents().strip()
    comments.append({'text': text, 'rating': rating, 'author': author})
```

В этом скрипте информация о комментариях (автор, рейтинг комментария и текст комментария) сохраняются в массиве словарей с ключами **'text'**, **'rating'** и **'author'**. Кроме того, при получении рейтинга извлечённое из страницы значение, которое является строкой, мы приводим к целочисленному типу с помощью функции **int()**.

Код, приведенный ниже, отображает сводную информацию по комментариям:

```
# вывод информации по комментариям
print('Комментариев в статье: ', len(comments))
print('Самый маленький комментарий:', sorted(comments, key=lambda x: len(x['text']))[0]['text'])
most_popular = sorted(comments, key=lambda x: x['rating'], reverse=True)[0]
print('Самый популярный комментарий:', most_popular['text'], 'Рейтинг ', most_popular['rating'])

# самый активный комментатор
commentators = {}
for comment in comments:
    if comment['author'] in commentators:
        commentators[comment['author']] += 1
    else:
        commentators[comment['author']] = 1
most_active = max(commentators, key=commentators.get)
print('Самый активный:', most_active, ', комментариев:', commentators[most_active])
```

В этом примере использовали ещё одну интересную возможность Python: лямбда-функции, например:

```
sorted(comments, key=lambda x: len(x['text']))
```

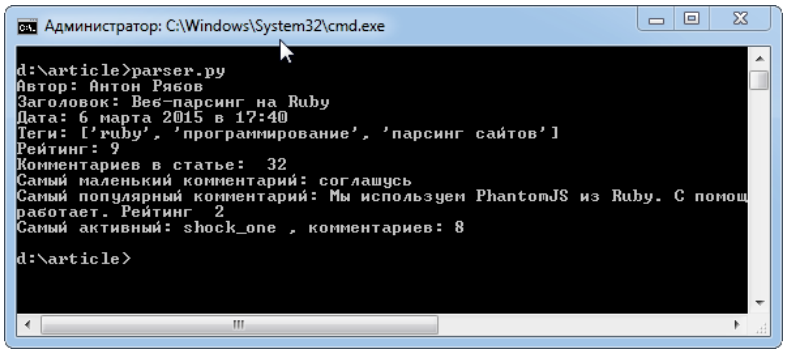
Лямбда-функции – это небольшие безымянные, встраиваемые прямо в код, функции, предназначенные для реализации специфического поведения, сохраняя при этом простоту и выразительность кода. В данном случае лямбда-функция позволяет выполнить сортировку комментариев по возрастанию длины текста в них.

Соберем все вместе. Итоговый скрипт приведен ниже.

```
from bs4 import BeautifulSoup as bs
import codecs

# открываем документ
doc = bs(codecs.open('article.htm', encoding='utf-8', mode='r').read(), 'html.parser')

# извлечение данных из статьи
author = doc.select('.author-info__name')[0].decode_contents().strip()
title = doc.select('.post__title span')[1].decode_contents().strip()
date = doc.select('.post__time_published')[0].decode_contents().strip()
tags = list(map(lambda x: x.decode_contents().strip(), doc.select('div.post__tags ul li a')))
```

```
rating = int(doc.select('div.post-additionals span.voting-wjt__counter-score.js-
score')[0].decode_contents().strip())

# вывод на экран
print('Автор:', author)
print('Заголовок:', title)
print('Дата:', date)
print('Теги:', tags)
print('Рейтинг:', rating)

# извлечение данных о комментариях
comments = []
for node in doc.select('div.comment_body'):
    text = node.select('div.message')[0].decode_contents().strip()
    rating = int(node.select('span.voting-wjt__counter-score.js-
score')[0].decode_contents().strip())
    author = node.select('a.comment-item__username')[0].decode_contents().strip()
    comments.append({'text': text, 'rating': rating, 'author': author})

# вывод информации по комментариям
print('Комментариев в статье: ', len(comments))
print('Самый маленький комментарий:', sorted(comments, key=lambda x: len(x['text']))[0]['text'])
most_popular = sorted(comments, key=lambda x: x['rating'], reverse=True)[0]
print('Самый популярный комментарий:', most_popular['text'], 'Рейтинг ', most_popular['rating'])

# самый активный комментатор
commentators = {}
for comment in comments:
    if comment['author'] in commentators:
        commentators[comment['author']] += 1
    else:
        commentators[comment['author']] = 1
most_active = max(commentators, key=commentators.get)
print('Самый активный:', most_active, ', комментариев:', commentators[most_active])
```

Результат работы скрипта:



Задание на практику

Требуется написать скрипт, выполняющий парсинг web-страницы. Страницу можно выбрать любую, но **сайты среди студентов повторяться не должны**.

Собранные данные необходимо отобразить на экране. Кроме того, нужно посчитать простейшие характеристики по собранным данным. Например,

- самый дешевый товар на странице;
- средняя цена на товар на странице;
- самый активный комментатор;
- количество уникальных пользователей в обсуждении.