**Модуль 10. Практическая работа (4 ак. ч.)**

**Вариант 3 Web Scraping с помощью python.**

Введение

Недавно заглянув на КиноПоиск, я обнаружил, что за долгие годы успел оставить более 1000 оценок и подумал, что было бы интересно поисследовать эти данные подробнее: менялись ли мои вкусы в кино с течением времени? есть ли годовая/недельная сезонность в активности? коррелируют ли мои оценки с рейтингом КиноПоиска, IMDb или кинокритиков?  
Но прежде чем анализировать и строить красивые графики, нужно получить данные. К сожалению, многие сервисы (и КиноПоиск не исключение) не имеют публичного API, так что, приходится засучить рукава и парсить html-страницы.

Именно о том, как скачать и распарсить web-cайт с помощью Web Scrapping,

[Новый Кинопоиск](http://plus.kinopoisk.ru/) под капотом использует запросы, которые возвращают данные об оценках в виде JSON, так что, задача могла быть решена и другим путем.

Задача

Задача будет состоять в том, чтобы выгрузить данные о просмотренных фильмах на КиноПоиске: название фильма (русское, английское), дату и время просмотра, оценку пользователя.  
На самом деле, можно разбить работу на 2 этапа:

* **Этап 1:** выгрузить и сохранить html-страницы
* **Этап 2:** распарсить html в удобный для дальнейшего анализа формат (csv, json, pandas dataframe etc.)

Инструменты

Для отправки http-запросов есть немало python-библиотек, наиболее известные urllib/urllib2 и Requests. На мой вкус [Requests](http://docs.python-requests.org/en/master/) удобнее и лаконичнее, так что, буду использовать ее.  
Также необходимо выбрать библиотеку для парсинга html, небольшой research дает следующие варианты:

* [**re**](https://docs.python.org/2/library/re.html)  
  Регулярные выражения, конечно, нам пригодятся, но использовать только их, на мой взгляд, слишком хардкорный путь, и они [немного не для этого](http://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags). Были придуманы более удобные инструменты для разбора html, так что перейдем к ним.
* [**BeatifulSoup**](http://www.crummy.com/software/BeautifulSoup/bs4/doc/), **[lxml](http://lxml.de/index.html)**  
  Это две наиболее популярные библиотеки для парсинга html и выбор одной из них, скорее, обусловлен личными предпочтениями. Более того, эти библиотеки тесно переплелись: BeautifulSoup стал использовать lxml в качестве внутреннего парсера для ускорения, а в lxml был добавлен модуль soupparser. Подробнее про плюсы и минусы этих библиотек можно почитать [в обсуждении](http://stackoverflow.com/questions/1922032/parsing-html-in-python-lxml-or-beautifulsoup-which-of-these-is-better-for-wha). Для сравнения подходов я буду парсить данные с помощью BeautifulSoup и используя [XPath](http://lxml.de/xpathxslt.html) селекторы в модуле lxml.html.
* [**scrapy**](http://scrapy.org/)  
  Это уже не просто библиотека, а целый open-source framework для получения данных с веб-страниц. В нем есть множество полезных функций: асинхронные запросы, возможность использовать XPath и CSS селекторы для обработки данных, удобная работа с кодировками и многое другое (подробнее можно почитать [тут](http://doc.scrapy.org/en/latest/intro/overview.html)). Если бы моя задача была не разовой выгрузкой, а production процессом, то я бы выбрала его. В текущей постановке это overkill.

Загрузка данных

Первая попытка

Приступим к выгрузке данных. Для начала, попробуем просто получить страницу по url и сохранить в локальный файл.

**import** requests

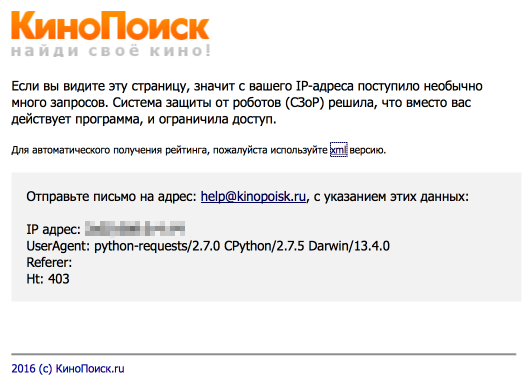
user\_id = 12345

url = 'http://www.kinopoisk.ru/user/%d/votes/list/ord/date/page/2/#list' % (user\_id) # url для второй страницы

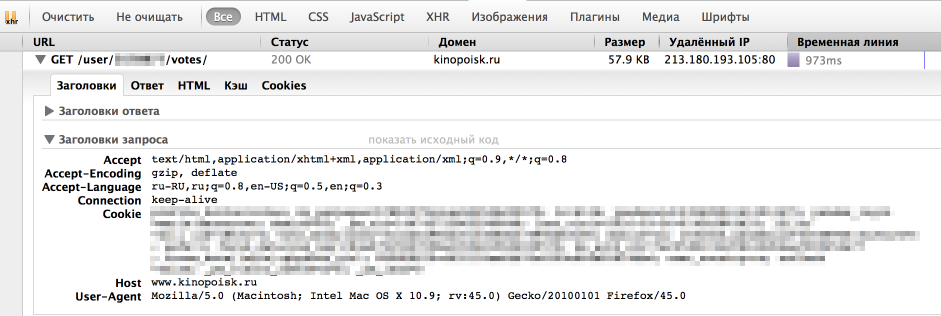
r = requests.get(url)

**with** open('test.html', 'w') **as** output\_file:

output\_file.write(r.text.encode('cp1251'))

Открываем полученный файл и видим, что все не так просто: сайт распознал в нас робота и не спешит показывать данные.  


Разберемся, как работает браузер

Однако, у браузера отлично получается получать информацию с сайта. Посмотрим, как именно он отправляет запрос. Для этого воспользуемся панелью "Сеть" в "Инструментах разработчика" в браузере (я использую для этого Firebug), обычно нужный нам запрос — самый продолжительный.  
  
  
  
Как мы видим, браузер также передает в headers UserAgent, cookie и еще ряд параметров. Для начала попробуем просто передать в header корректный UserAgent.

headers = {

'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:45.0) Gecko/20100101 Firefox/45.0'

}

r = requests.get(url, headers = headers)

На этот раз все получилось, теперь нам отдаются нужные данные. Стоит отметить, что иногда сайт также проверяет корректность cookie, в таком случае помогут [sessions](http://docs.python-requests.org/en/master/user/advanced/) в библиотеке Requests.

Скачаем все оценки

Теперь мы умеем сохранять одну страницу с оценками. Но обычно у пользователя достаточно много оценок и нужно проитерироваться по всем страницам. Интересующий нас номер страницы легко передать непосредственно в url. Остается только вопрос: "Как понять сколько всего страниц с оценками?" Я решила эту проблему следующим образом: если указать слишком большой номер страницы, то нам вернется вот такая страница без таблицы с фильмами. Таким образом мы можем итерироваться по страницам до тех, пор пока находится блок с оценками фильмов (<div class = "profileFilmsList">).  
  


Полный код для загрузки данных

Парсинг

Немного про XPath

XPath — это язык запросов к xml и xhtml документов. Мы будем использовать XPath селекторы при работе с библиотекой lxml ([документация](http://lxml.de/xpathxslt.html)). Рассмотрим небольшой пример работы с XPath

**from** lxml **import** html

test = '''

<html>

<body>

<div class="first\_level">

<h2 align='center'>one</h2>

<h2 align='left'>two</h2>

</div>

<h2>another tag</h2>

</body>

</html>

'''

tree = html.fromstring(test)

tree.xpath('//h2') # все h2 теги

tree.xpath('//h2[@align]') # h2 теги с атрибутом align

tree.xpath('//h2[@align="center"]') # h2 теги с атрибутом align равным "center"

div\_node = tree.xpath('//div')[0] # div тег

div\_node.xpath('.//h2') # все h2 теги, которые являются дочерними div ноде

Подробнее про синтаксис XPath также можно почитать на [W3Schools](http://www.w3schools.com/xsl/xpath_intro.asp).

Вернемся к нашей задаче

Теперь перейдем непосредственно к получению данных из html. Проще всего понять как устроена html-страница используя функцию "Инспектировать элемент" в браузере. В данном случае все довольно просто: вся таблица с оценками заключена в теге <div class = "profileFilmsList">. Выделим эту ноду:

**from** bs4 **import** BeautifulSoup

**from** lxml **import** html

# Beautiful Soup

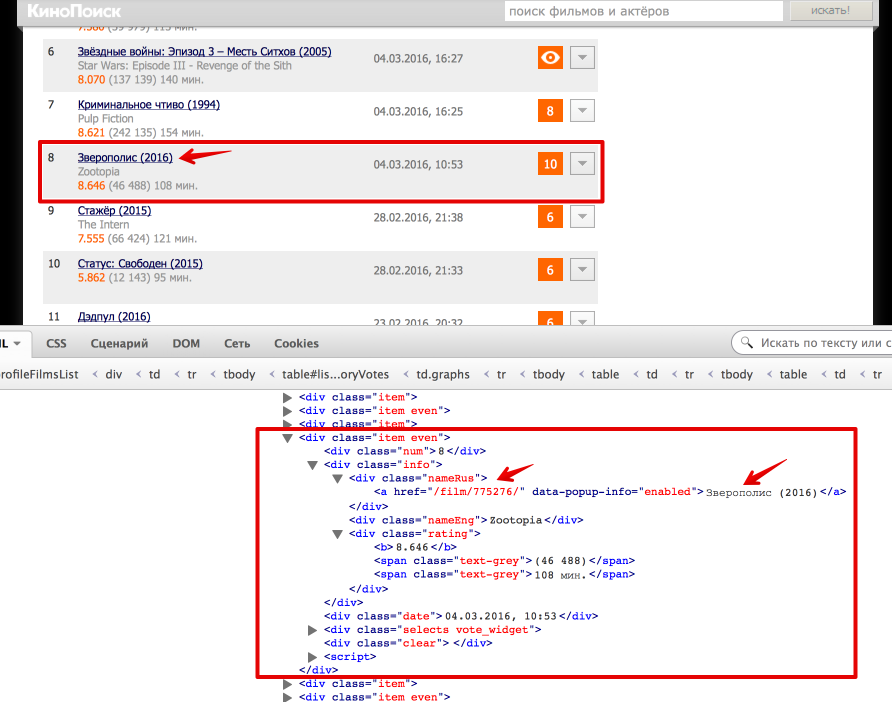
soup = BeautifulSoup(text)

film\_list = soup.find('div', {'class': 'profileFilmsList'})

# lxml

tree = html.fromstring(text)

film\_list\_lxml = tree.xpath('//div[@class = "profileFilmsList"]')[0]

Каждый фильм представлен как <div class = "item"> или <div class = "item even">. Рассмотрим, как вытащить русское название фильма и ссылку на страницу фильма (также узнаем, как получить текст и значение атрибута).  


# Beatiful Soup

movie\_link = item.find('div', {'class': 'nameRus'}).find('a').get('href')

movie\_desc = item.find('div', {'class': 'nameRus'}).find('a').text

# lxml

movie\_link = item\_lxml.xpath('.//div[@class = "nameRus"]/a/@href')[0]

movie\_desc = item\_lxml.xpath('.//div[@class = "nameRus"]/a/text()')[0]

Еще небольшой хинт для debug'a: для того, чтобы посмотреть, что внутри выбранной ноды в BeautifulSoup можно просто распечатать ее, а в lxml воспользоваться функцией tostring() модуля etree.

# BeatifulSoup

print item

#lxml

**from** lxml **import** etree

print etree.tostring(item\_lxml)

Полный код для парсинга html-файлов под катом

**def** **read\_file**(filename):

**with** open(filename) **as** input\_file:

text = input\_file.read()

**return** text

**def** **parse\_user\_datafile\_bs**(filename):

results = []

text = read\_file(filename)

soup = BeautifulSoup(text)

film\_list = film\_list = soup.find('div', {'class': 'profileFilmsList'})

items = film\_list.find\_all('div', {'class': ['item', 'item even']})

**for** item **in** items:

# getting movie\_id

movie\_link = item.find('div', {'class': 'nameRus'}).find('a').get('href')

movie\_desc = item.find('div', {'class': 'nameRus'}).find('a').text

movie\_id = re.findall('\d+', movie\_link)[0]

# getting english name

name\_eng = item.find('div', {'class': 'nameEng'}).text

#getting watch time

watch\_datetime = item.find('div', {'class': 'date'}).text

date\_watched, time\_watched = re.match('(\d{2}\.\d{2}\.\d{4}), (\d{2}:\d{2})', watch\_datetime).groups()

# getting user rating

user\_rating = item.find('div', {'class': 'vote'}).text

**if** user\_rating:

user\_rating = int(user\_rating)

results.append({

'movie\_id': movie\_id,

'name\_eng': name\_eng,

'date\_watched': date\_watched,

'time\_watched': time\_watched,

'user\_rating': user\_rating,

'movie\_desc': movie\_desc

})

**return** results

**def** **parse\_user\_datafile\_lxml**(filename):

results = []

text = read\_file(filename)

tree = html.fromstring(text)

film\_list\_lxml = tree.xpath('//div[@class = "profileFilmsList"]')[0]

items\_lxml = film\_list\_lxml.xpath('//div[@class = "item even" or @class = "item"]')

**for** item\_lxml **in** items\_lxml:

# getting movie id

movie\_link = item\_lxml.xpath('.//div[@class = "nameRus"]/a/@href')[0]

movie\_desc = item\_lxml.xpath('.//div[@class = "nameRus"]/a/text()')[0]

movie\_id = re.findall('\d+', movie\_link)[0]

# getting english name

name\_eng = item\_lxml.xpath('.//div[@class = "nameEng"]/text()')[0]

# getting watch time

watch\_datetime = item\_lxml.xpath('.//div[@class = "date"]/text()')[0]

date\_watched, time\_watched = re.match('(\d{2}\.\d{2}\.\d{4}), (\d{2}:\d{2})', watch\_datetime).groups()

# getting user rating

user\_rating = item\_lxml.xpath('.//div[@class = "vote"]/text()')

**if** user\_rating:

user\_rating = int(user\_rating[0])

results.append({

'movie\_id': movie\_id,

'name\_eng': name\_eng,

'date\_watched': date\_watched,

'time\_watched': time\_watched,

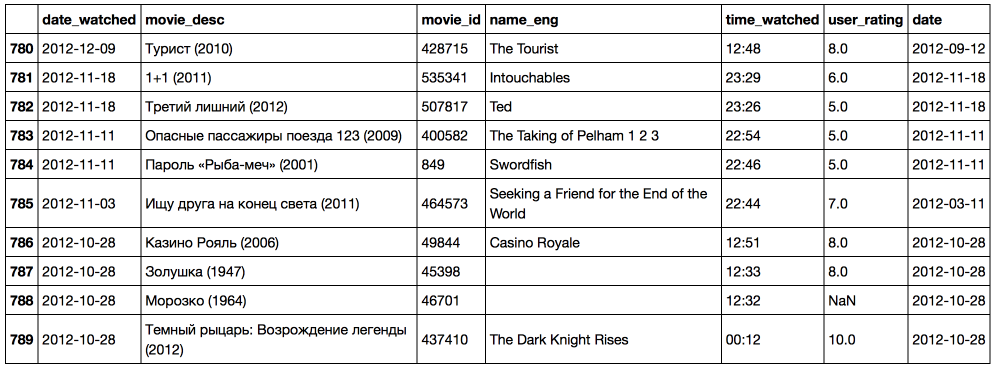
'user\_rating': user\_rating,

'movie\_desc': movie\_desc

})

**return** results

Резюме

В результате, мы научились парсить web-сайты, познакомились с библиотеками Requests, BeautifulSoup и lxml, а также получили пригодные для дальнейшего анализа данные о просмотренных фильмах на КиноПоиске.  
  
Полный код проекта можно найти на [github'e](https://github.com/miptgirl/kinopoisk_data/blob/master/loading_kp_data.ipynb).

UPD

Как отметили в [комментариях](https://habrahabr.ru/post/280238/#comment_8823694), в контексте Web Scrapping'a могут оказаться полезны следующие темы:

* **Аутентификация:** зачастую для того, чтобы получить данные с сайта нужно пройти аутентификацию, в простейшем случае это просто HTTP Basic Auth: логин и пароль. Тут нам снова [поможет](http://docs.python-requests.org/en/master/user/authentication/) библиотека Requests. Кроме того, широко распространена oauth2: как использовать oauth2 в python можно почитать на [stackoverflow](http://stackoverflow.com/questions/9548729/how-to-authenticate-a-site-with-python-using-urllib2). Также в [комментариях](https://habrahabr.ru/post/280238/#comment_8824194) есть пример от [Terras](https://habrahabr.ru/users/terras/) того, как пройти аутентификацию в web-форме.
* **Контролы:** На сайте также могут быть дополнительные web-формы (выпадающие списки, check box'ы итд). Алгоритм работы с ними примерно тот же: смотрим, что посылает браузер и отправляем эти же параметры как data в POST-запрос ([Requests](http://docs.python-requests.org/en/master/user/quickstart/" \l "more-complicated-post-requests), [stackoverflow](http://stackoverflow.com/questions/8377055/submit-data-via-web-form-and-extract-the-results)). Также могу порекомендовать посмотреть 2й урок [курса "Data Wrangling" на Udacity](https://www.udacity.com/course/data-wrangling-with-mongodb--ud032), где подробно рассмотрен пример scrapping сайта [US Department of Transportation](http://www.transtats.bts.gov/Data_Elements.aspx?Data=2) и посылка данных web-форм.