GeekBrains

Методы сбора и обработки данных при помощи Python

Основы клиент-серверного взаимодействия. Парсинг API

На этом уроке

1. Узнаем основные принципы сбора данных.
2. Как отправлять GET-запросы при помощи разных инструментов.
3. Как работать с ответами от сервера и API, JSON

Оглавление

При написании руководства были использованы ОС: Windows 10 x64, Python v3.7.2, PyCharm Edu v2019.1.1, Postman v7.5.0, ConEmu v161206

Вам предстоит работа с командной строкой. Если вы используете ОС Windows, рекомендуется установить более удобный аналог командной строки ConEmu.

Парсинг, скрапинг, краулинг

Всеобщие глобальные тенденции приближают нас к тому дню, когда все операции и торговые сделки будут проходить в интернете. Чтобы успешно вписаться в новый порядок, очень важно своевременно получать актуальные данные о движении рынка (динамика цен и товаров) и локальные новости, которые часто влияют на формирование спроса.

Сегодня объёмы информации превосходят возможности их обработки у любого, даже самого талантливого человека или узкопрофильного специалиста. Поэтому для автоматического сбора и обработки больших объёмов информации было придумано множество методов, знание и умелое использование которых — суть новой профессии Data Scientist.

Парсинг

Парсинг — это синтаксический анализ информации. Под парсингом HTML, как правило, подразумевают выборочное извлечение большого количества информации с других сайтов и ее последующее использование.

Парсить сайт — значит искать на странице при помощи специальных инструментов необходимый участок кода, в котором заключена нужная нам информация. Эти части кода повторяются в пределах одной страницы и на других страницах, имеющих аналогичную структуру. Парсер помогает нам выделить и импортировать повторяющиеся данные автоматически, существенно сэкономив время и предупредив возможные ошибки копирования этой информации вручную.

Скрапинг

В широком понимании веб-скрапинг — это сбор данных с различных интернет-ресурсов. Общий принцип его работы можно объяснить следующим образом: автоматизированный код выполняет GET-запросы на целевой сайт и, получая ответ, парсит HTML-документ, ищет данные и преобразует их в заданный формат. Заметим, что к категории полезных данных могут относиться:

- каталоги товаров;
- изображения;
- видео;
- текстовый контент;
- открытые контактные данные — адреса электронной почты, телефоны и т. д.

Существует масса решений для скрапинга веб-сайтов. Среди них:

- отдельные сервисы, которые работают через API или имеют веб-интерфейс (Embedly, DiffBot и др.);
- проекты на разных языках программирования с открытым кодом (Goose, Scrapy — Python; Goutte — PHP; Readability, Morph — Ruby).

Кроме того, всегда есть возможность изобрести велосипед и написать собственное решение, например, с использованием библиотеки lxml (для языка программирования Python).

Однако следует понимать, что идеального скрапера не существует, потому что:

1. Сайтов, свёрстанных строго по канонам веб-дизайна, не существует (но в этом залог их своеобразия и привлекательности для пользователей).

2. Каждый веб-разработчик (если он не работает в солидной IT-компании со своими правилами и стайлгайдами) пишет код под себя или просто как умеет. Далеко не всегда код получается грамотным и качественным. Ошибки в «самописном» коде делают его нечитаемым для скраперов (речь в первую очередь идет о вёрстке).
3. Масса веб-ресурсов использует HTML5, где каждый элемент может быть абсолютно уникальным.
4. Некоторые ресурсы защищены от копирования данных (многоуровневая верстка, использование JavaScript для рендеринга контента, проверка user agent и т. д.), а значит, и от скрапинга.
5. В зависимости от сезона или тематики целевого материала, на сайте могут использоваться разные макеты. Периодически это касается даже типичных страниц (сезонные акции, премиум-статьи и т. д.).
6. Веб-страницы часто изобилуют «мусором» (реклама, комментарии, дополнительные элементы навигации и т. д.).
7. Исходный код может содержать ссылки на одни и те же картинки разных размеров, например для превью.
8. Сайт может определить страну, в которой находится ваш сервер, и отдать информацию не на английском языке.
9. У всех сайтов может быть разная кодировка, которая не отдаётся в ответе на запрос.

Все вышеперечисленное серьезно затрудняет веб-скрапинг. В результате качество контента может упасть до 20% и даже до 10%, что абсолютно неприемлемо. Здесь можно руководствоваться двумя правилами:

1. При необходимости получать данные из небольшого количества источников. Лучше написать свой скрапер и настроить его под нужные сайты (качество получаемого контента — около 100%).
2. Использовать комплексный подход в выборе ридера (скрапера), если нужно получать информацию из большого количества источников (до 95% качества).

Краулинг

Краулинг (сканирование, crawling) — процесс обнаружения и сбора поисковым роботом (краулером) новых и обновлённых страниц для добавления в индекс поисковых систем. Сканирование — начальный этап: данные собираются только для дальнейшей внутренней обработки (построения индекса) и не отображаются в результатах поиска. Просканированная страница не всегда оказывается проиндексированной.

Поисковый робот (он же crawler, краулер, паук, бот) — программа для сбора контента в интернете. Краулер состоит из множества компьютеров, запрашивающих и выбирающих страницы намного быстрее, чем пользователь с помощью веб-браузера. Фактически он может запрашивать тысячи разных страниц одновременно.

Выдача ответов на поисковый запрос на странице поиска за долю секунды — только верхушка айсберга. В «чёрном ящике» поисковых систем — просканированные и занесенные в специальную базу данных миллиарды страниц, которые отбираются для представления с учётом множества факторов.

Страница с результатами поиска формируется в результате трёх процессов:

- сканирования;
- индексирования;

- предоставления результатов (состоит из поиска по индексу и ранжирования страниц).

Что ещё делает робот-краулер:

- постоянно проверяет и сравнивает список URL-адресов для сканирования с URL-адресами, которые уже находятся в индексе;
- убирает дубликаты в очереди, чтобы предотвратить повторное скачивание одной и той же страницы;
- добавляет на переиндексацию измененные страницы для предоставления обновленных результатов.

При сканировании пауки просматривают страницы и выполняют переход по содержащимся на них ссылкам так же, как и обычные пользователи. При этом разный контент исследуется ботами в разной последовательности. Это позволяет одновременно обрабатывать огромные массивы данных.

Например, в Google существуют роботы для обработки разного типа контента:

- Googlebot — основной поисковый робот;
- Googlebot News — робот для сканирования новостей;
- Googlebot Images — робот для сканирования изображений;
- Googlebot Video — робот для сканирования видео.

HTTP и HTTPS

HyperText Transfer Protocol — протокол, созданный для передачи гипертекстовых документов. Сейчас по нему можно передавать произвольную информацию. HTTP — протокол прикладного слоя модели OSI. Он используется для приложений и пользователей.

HTTPS — это расширение протокола HTTP с поддержкой шифрования для повышения безопасности. Данные по HTTPS передаются поверх протокола TLS. Протокол TLS — криптографический и служит для защищённой передачи данных в Интернете.

HTTP-коды

При отправке любого запроса на сервер (в том числе когда вы просто открываете страницу сайта в браузере) в любом случае возвращается ответ на ваш запрос в виде кода. Думаю, каждый сталкивался с известной ошибкой 404.

Код состояния HTTP — часть первой строки ответа сервера при запросах по протоколу HTTP. Он представляет собой целое число из трёх десятичных цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа. Примеры:

- 201 Created;
- 401 Unauthorized;
- 507 Insufficient Storage.

Клиент может не знать все коды состояния, но он обязан отреагировать в соответствии с классом кода. Выделено пять классов кодов состояния:

1. **1xx: Information.**

В этот класс выделены коды, информирующие о процессе передачи. Это обычно предварительный ответ, который состоит только из Status Line и опциональных заголовков и завершается пустой строкой. Обязательных заголовков для этого класса кодов состояния нет. Из-за того, что стандарт протокола HTTP/1.0 не определял информационных кодов состояния, серверы **не обязаны** посылать 1xx-ответы HTTP/1.0-клиентам, за исключением экспериментальных условий.

2. **2xx: Success.**

Этот класс кодов состояния указывает, что запрос клиента был успешно получен, понят, и принят. Примеры ответов: 200 OK, 201 Created.

3. **3xx: Redirect.**

Класс кодов состояния показывает, какие действия должен выполнить клиент, чтобы запрос завершился успешно. Служит для переадресации на мобильную версию, на HTTPS, когда клиент подключается по HTTP, или на другое доменное имя, если сайт переехал или доступен по нескольким доменным именам.

4. **4xx: Client Error.**

Класс кодов 4xx предназначен для определения ошибок, которые произошли на стороне клиента. К примеру, ошибка 404 Not Found возникает, когда клиент обращается к ресурсу, которого не существует.

5. **5xx: Server Error.**

Коды ответов, начинающиеся с 5, указывают на случаи, когда сервер знает, что произошла ошибка, или не может обработать запрос. Кроме HEAD-запросов, серверу следует включить в ответ сущность, содержащую объяснение ошибки, и указать, постоянное это состояние или временное. Агентам **следует** показывать включенную сущность пользователям. Этот код ответа подходит для любых методов запросов. Примеры: ошибка 500 Internal Server Error возникает когда сервер недоступен, а 502 Bad Gateway — когда балансировщик, например Nginx, выступает в роли прокси, предоставляет доступ к другому ресурсу и получил ошибку от целевого ресурса (бэкенда), которую не смог обработать.

HTTP-методы

HTTP-метод — это указание на основную операцию над ресурсом, то есть команда для сервера. Обычно это короткое слово, написанное заглавными буквами. Основные методы:

- GET — используется для запроса содержимого указанного ресурса;
- PUT — применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существует ресурс, то сервер создает его и возвращает статус 201 (Created). Если же ресурс был изменён, сервер возвращает 200 (Ok) или 204 (No Content).
- POST — применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы на сервер.
- DELETE — удаляет указанный ресурс.
- PATCH — аналогично PUT, но применяется только к фрагменту ресурса.

Есть и другие методы — HEAD, OPTIONS, TRACE, CONNECT, но они применяются очень редко и крайне специфичны.

Важно!

*Спецификация HTTP **не обязывает** сервер понимать все методы, а также **не указывает**, что ему делать и как реагировать на методы. Поэтому запрос к серверу методом GET может вместо того, чтобы запросить ресурс, удалить его.*

HTTP-заголовки

Заголовки HTTP — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару имя-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA.

Все заголовки разделяются на четыре основных группы:

- General Headers — должны включаться в любое сообщение клиента и сервера. Пример: **User-Agent**, который указывает программное обеспечение клиента и его характеристики.
- Request Headers — используются только в запросах клиента. Пример: **Authorization**, в котором передаются данные для авторизации.
- Response Headers — только для ответов от сервера. Пример: **Age**, который хранит количество секунд с момента модификации ресурса.
- Entity Headers — сопровождают каждую сущность сообщения, используются при передаче множественного содержимого (multipart/*). Такие заголовки характеризуют каждый конкретный блок информации: например, ответ сервера содержит как JSON-информацию, так и файл, разбитый на части. Например: **Allow**, в котором содержится список поддерживаемых методов

Именно в таком порядке рекомендуется посылать заголовки получателю.

GET-запросы

Наша цель — получение информации, поэтому мы будем работать в основном с GET-запросами. Рассмотрим подробнее, как и какими способами можно их выполнять.

Согласно стандарту HTTP, запросы типа GET считаются идемпотентными, т. е., при повторных выполнениях сервер будет возвращать один и тот же результат.

GET-запросы с помощью cURL

cURL — кроссплатформенная служебная программа командной строки, позволяющая взаимодействовать со множеством различных серверов по множеству различных протоколов с синтаксисом URL.

Синтаксис утилиты очень прост:

curl опции ссылка

Основные опции:

- **-#** — отображать простой прогресс-бар во время загрузки;
- **-0** — использовать протокол http 1.0;
- **-1** — использовать протокол шифрования tlsv1;

- **-2** — использовать sslv2;
- **-3** — использовать sslv3;
- **-4** — использовать ipv4;
- **-6** — использовать ipv6;
- **-A** — указать свой USER_AGENT;
- **-b** — сохранить cookie в файл;
- **-c** — отправить cookie на сервер из файла;
- **-C** — продолжить загрузку файла с места разрыва или указанного смещения;
- **-m** — максимальное время ожидания ответа от сервера;
- **-d** — отправить данные методом POST;
- **-D** — сохранить заголовки, возвращенные сервером в файл;
- **-e** — задать поле Referer URI, которое указывает, с какого сайта пришел пользователь;
- **-E** — использовать внешний сертификат SSL;
- **-f** — не выводить сообщения об ошибках;
- **-F** — отправить данные в виде формы;
- **-G** — если эта опция включена, то все данные, указанные в опции -d, будут передаваться методом GET;
- **-H** — передать заголовки на сервер;
- **-I** — получить только HTTP-заголовок, а всё содержимое страницы проигнорировать;
- **-j** — прочитать и отправить cookie из файла;
- **-J** — удалить заголовок из запроса;
- **-L** — принимать и обрабатывать перенаправления;
- **-s** — максимальное количество перенаправлений с помощью Location;
- **-o** — выводить контент страницы в файл;
- **-O** — сохранять контент в файл с именем страницы или файла на сервере;
- **-p** — использовать прокси;
- **--proto** — указать протокол, который нужно использовать;
- **-R** — сохранять время последнего изменения удалённого файла;
- **-s** — выводить минимум информации об ошибках;
- **-S** — выводить сообщения об ошибках;
- **-T** — загрузить файл на сервер;
- **-u —** передать логин и пароль для авторизации в виде username:password;
- **-v** — максимально подробный вывод;
- **-y** — минимальная скорость загрузки;
- **-Y** — максимальная скорость загрузки;
- **-z** — скачать файл, только если он был модифицирован позже указанного времени;
- **-V** — вывести версию.

Пример запроса на адрес http://google.com при помощи утилиты curl:

```
curl -v http://google.com
```

Ответ на запрос:

```
* Rebuilt URL to: http://google.com/
*   Trying 209.85.233.113...
* TCP_NODELAY set
* Connected to google.com (209.85.233.113) port 80 (#0)
> GET / HTTP/1.1
> Host: google.com
> User-Agent: curl/7.55.1
> Accept: */*
>
```

```
< HTTP/1.1 301 Moved Permanently
< Location: http://www.google.com/
< Content-Type: text/html; charset=UTF-8
< Date: Sun, 25 Aug 2019 11:23:44 GMT
< Expires: Tue, 24 Sep 2019 11:23:44 GMT
< Cache-Control: public, max-age=2592000
< Server: gws
< Content-Length: 219
< X-XSS-Protection: 0
< X-Frame-Options: SAMEORIGIN
<
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">hereOK</A>.
</BODY></HTML>
* Connection #0 to host google.com left intact
```

Вывод содержит несколько блоков важной информации:

- строки, начинающиеся с символа *****, — это **системные сообщения**, генерируемые ОС;
- строки, начинающиеся с символа **>**, — это **строки запроса** к серверу;
- строки, начинающиеся с символа **<**, — **строки ответа** от сервера;
- Строка GET / HTTP/1.1 — это строка запроса;
- Строки **Host: google.com**, **User-Agent: curl/7.55.1**, **Accept: */*** — это **заголовки**.

В конце после пустой строки — содержимое ответа от сервера. В случае с обычным запросом на сайт возвращается HTML-код страницы.

Важно!

Если при запросе по протоколу HTTPS вы получаете ответ вида:
curl: (35) schannel: next InitializeSecurityContext failed: Unknown error (0x80092012) – Функция отзыва не смогла произвести проверку отзыва для сертификата,
приостановите работу вашей антивирусной программы.

HTTP-заголовков очень много, и все они бывают нужны для разных задач, для взаимодействия с разными приложениями. Одна из таких задач — защита веб-сервера от «непрошеных гостей». Самый простой способ защиты — Basic Authorization (BA). При неавторизованном запросе к защищенному ресурсу:

```
curl -v https://postman-echo.com/basic-auth
```

Ответ будет выглядеть следующим образом:

```
< HTTP/1.1 401 Unauthorized
< Date: Sun, 25 Aug 2019 11:43:29 GMT
< Server: nginx
< set-cookie:
sails.sid=s%3AxFs-Tao2S3ixv6IVLLU_ijWmIKfr4Vxm.oAHGFs4OQUcoHRYmC4QsETXsDISEeJUBy
NhQHdNLzqU; Path=/; HttpOnly
< WWW-Authenticate: Basic realm="Users"
< Content-Length: 12
```

```
< Connection: keep-alive
```

Мы видим HTTP-код 401 — статус ответа, который говорит нам, что требуется авторизация. Также мы видим заголовок WWW-Authenticate, который начинается со слов Basic realm, — значит, применена BA. Если выполнить такой запрос в браузере, то браузер, получив подобный ответ от сервера, покажет окошко для ввода логина и пароля. Если нажать «Отмена», повторного запроса не будет, потому что ответ от сервера уже получен. А вот если ввести данные и нажать ОК, то браузер сформирует ещё один запрос, но уже **добавит в него новый заголовок**:

```
Authorization: Basic <base64_string_here>
```

Логин и пароль передаются в открытом виде, base64 — это всего лишь закодированная строка, раскодировав которую стандартным алгоритмом без применения шифрования, мы получим строку в виде пары «логин/пароль», разделенной двоеточием.

Вот другой пример с передачей заголовка, который выдаст ваш запрос как выполняемый через веб-браузер Mozilla Firefox:

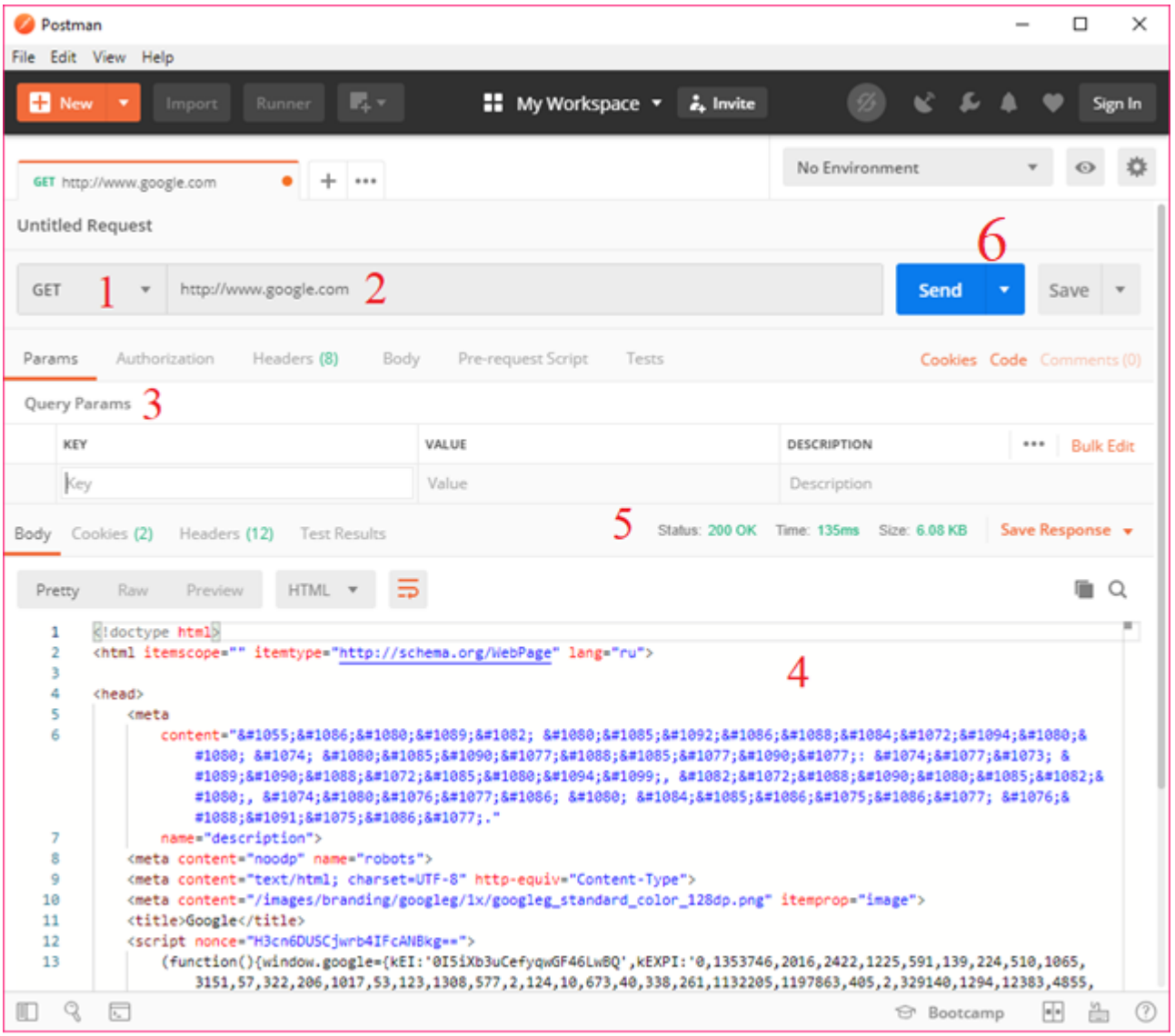
```
curl -v –H "User-Agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
https://postman-echo.com/basic-auth
```

При необходимости можно воспользоваться встроенной помощью утилиты. Вызывается она так:

```
curl --help
```

GET-запросы с помощью Postman

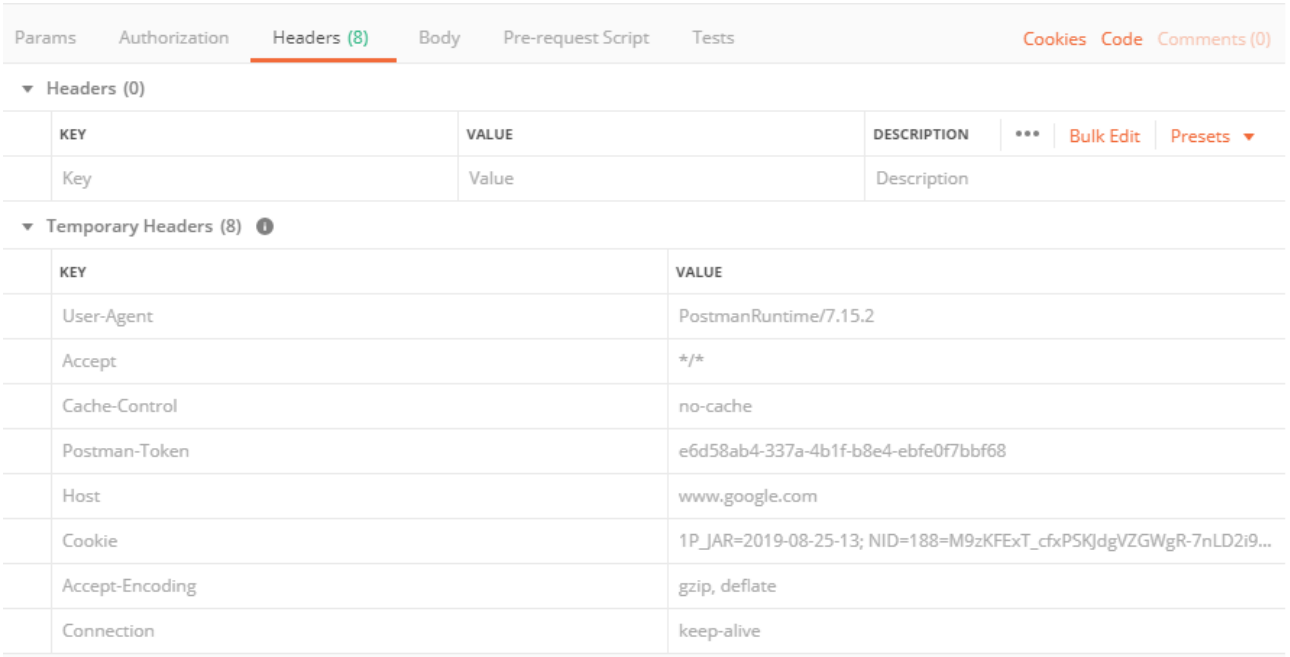
Основное предназначение приложения Postman — создание коллекций с запросами к API или веб-сервису. Оно позволяет формировать запросы гораздо более удобным способом, чем cURL, а также имеет графический интерфейс:



На рисунке:

1 — выпадающий список, который позволяет выбрать любой тип запроса;
2 — адрес сервиса, к которому отправляется запрос;
3 — параметры запроса;
4 — ответ от сервера;
5 — информация об ответе от сервера (код статуса, время ответа, размер ответа);
6 — кнопка отправки запроса на сервер.

Переключившись на закладку Headers, вы сможете указать заголовки для вашего запроса:



Раздел Headers содержит заголовки, указанные вручную.

Раздел Temporary Headers содержит заголовки, которые были сформированы самим приложением Postman.

Важно!

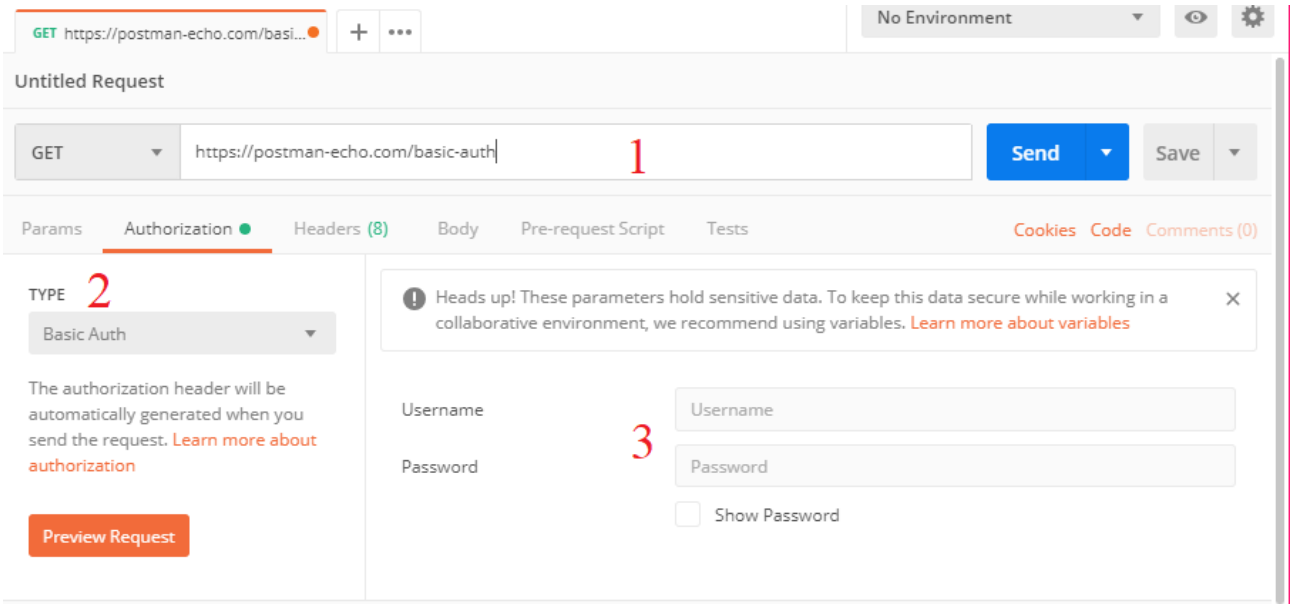
Чтобы избежать дополнительных проблем при запросах, лучше всего передавать вместе с вашим запросом заголовок, имитирующий запрос из реального браузера. Для этого передайте заголовок User-Agent:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) — для Mozilla Firefox,

AppleWebKit/537.36 (KHTML, like Gecko) — для Safari,

Chrome/76.0.3809.100 — для Google Chrome.

Postman удобен тем, что не обязательно изучать синтаксис команд — всё интуитивно понятно. Например, передать тот же самый запрос с авторизацией можно, указав параметры в соответствующей закладке Authorization:



На рисунке:

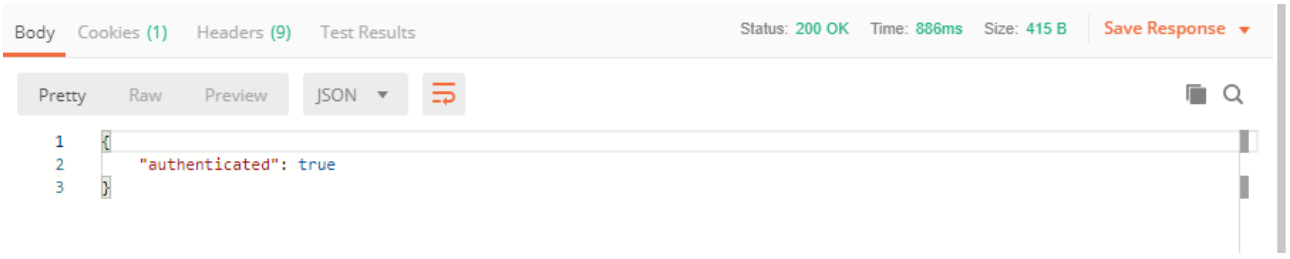
1 — сервис, к которому отправляем запрос;
2 — выпадающий список с разными типами авторизации;
3 — поля для ввода логина и пароля.

При нажатии на кнопку Send будет в

	KEY	VALUE
	Authorization	Basic cG9zdG1hbjpwYXNzd29yZA==

выполнен GET-запрос, и соответствующие заголовки формируются автоматически:

Как и в случае с cURL, имя пользователя и пароль передаются в виде закодированной строки. Соответственно, никто не мешает вам сформировать указанный заголовок вручную и передать его вместе с вашим запросом. Результат работы будет таким же:



GET-запросы в Python

Чтобы выполнять веб-запросы на любой интернет-ресурс, нам понадобится библиотека **requests**.

Requests — библиотека Python, которая элегантно и просто выполняет HTTP-запросы. Она отсутствует в стандартной поставке Python, и её необходимо предварительно скачать и установить:

```
pip install requests
```

Импортируем модуль себе стандартной командой:

```
import requests
```

С его помощью мы можем выполнить все стандартные запросы на сервер:

```
req = requests.put("http://httpbin.org/put")
req = requests.delete("http://httpbin.org/delete")
req = requests.head("http://httpbin.org/get")
req = requests.options("http://httpbin.org/get")
```

Результатом запроса будет объект **response**:

```
print(type(req))
```

```
<class 'requests.models.Response'>
```

Выполним GET-запрос на сайт Яндекса:

```
import requests
req = requests.get("https://yandex.ru")
```

Теперь **req** — объект **response.** Он содержит ответ от сервера на наш HTTP-запрос. Основные методы и свойства объекта **response**:

- **headers** — возвращает заголовки ответа от сервера;
- **status_code** — возвращает код состояния ответа от сервера;
- **ok** — возвращает **True**, если код состояния меньше 400, и **False**, если нет;
- **text** — возвращает содержимое ответа в Unicode (кодировке сайта);
- **json** — возвращает ответ от сервера в формате JSON;
- **content** — возвращает содержимое ответа в бинарном виде.

Посмотрим на содержимое ответа:

```
print('Заголовки: \n',  req.headers)
print('Ответ: \n',  req.text)
```

```
Заголовки:
 {'Content-Security-Policy': "connect-src 'self' wss://webasr.yandex.net
                                        .
                                        .
'Transfer-Encoding': 'chunked', 'Cache <!DOCTYPE html><html class="i-ua_js_no
i-ua_css_standart i-ua_browser_unknown i-ua_browser_desktop i-ua_platform_other"
lang="ru"><head xmlns:og="http://ogp.me/ns#"><meta http-equiv=Content-Type
content="text/html;charset=UTF-8">
-Control': 'no-cache,no-store,max-age=0,must-revalidate', 'Content-Encoding':
'gzip', 'Content-Type': 'text/html; charset=UTF-8'}
```

```
Ответ:
                                                .
                                                .
</div><!--/noindex--><!--vla1-9858-251-vla-portal-morda-31387.gencfg-c.yandex.ne
t--></body></html>
```

Как и в случае с cURL, заголовки здесь необходимо формировать вручную. Передаются они как именованные параметры при вызове метода GET объекта **requests**. Выполним запрос на сервис, использующий Basic Authorization, выдав себя за браузер:

```
import requests
headers = {'User-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)',
                    'Authorization':'Basic cG9zdG1hbjpwYXNzd29yZA==='}
req = requests.get('https://postman-echo.com/basic-auth',headers=headers)
print('Заголовки: \n',  req.headers)
print('Ответ: \n',  req.text)
```

Получим следующий результат:

```
Заголовки:
 {'Content-Encoding': 'gzip', 'Content-Type': 'application/json; charset=utf-8',
'Date': 'Sun, 25 Aug 2019 16:43:31 GMT', 'ETag':
'W/"16-sJz8uwjdDv0wvm7//BYdNw8vMbU"', 'Server': 'nginx', 'set-cookie':
'sails.sid=s%3AoWocE1e1MVuuc-Heg4tGwrvZ3Eb2U-kA.w2TqQA5TwkFiz475tyvCBrvrjstjmFCM
N3kY1IP9LL8; Path=/; HttpOnly', 'Vary': 'Accept-Encoding', 'Content-Length':
'42', 'Connection': 'keep-alive'}
Ответ:
 {"authenticated":true}
```

Мы получили тот же самый результат, что и при использовании утилит Postman и cURL.

API

Что такое API

Application Programming Interface (читается «эй-пи-ай») — программный интерфейс приложения. Строго говоря, API — это набор готовых классов, библиотек и программных интерфейсов, которые облегчают программисту написание кода.

Со стороны веба это выглядит так: есть сервис, которые собирает данные, строит по ним статистику, графики и прочее, делает отчеты. Этот публичный сервис упрощает сторонним разработчикам жизнь и создаёт свое API, которое позволяет быстро и правильно забирать эти данные для своих нужд. Про такие API мы и будем говорить дальше.

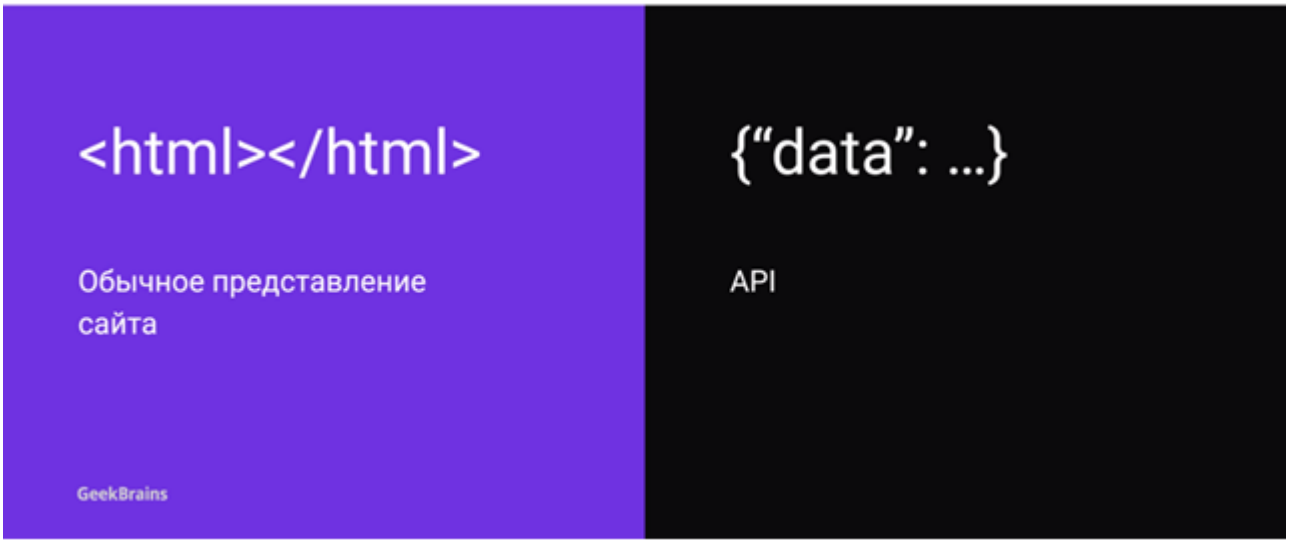
API бывают двух видов: публичные и приватные.

Публичный API отдаётся в использование другим людям и приложениям. Примеры таких API — свободные для скачивания библиотеки, а также API таких сервисов, как Twitter или Facebook,

позволяющие получить данные программным путем. Здесь очень важно отметить, что публичный API должен следить за обратной совместимостью: нельзя сначала включить в API метод получения всех данных без параметров, а при релизе следующей версии убрать его и сказать, что теперь его нет, есть другой метод, точно такой же, но он уже требует какой-нибудь параметр на вход.

Приватный API — это API, который создан для нужд системы. Вот пример: есть веб-сервис, который авторизует пользователей: пользователь вводит логин и пароль, а дальше происходят идентификация, аутентификация и авторизация. Эти процессы проходят не на самом веб-сервисе — он вызывает другой сервис через его API, передает логин и пароль (в зашифрованном виде), после чего ожидает получить от него ответ. Это яркий пример приватного API. Здесь можно не следить за обратной совместимостью, так как клиенты этого API вам известны, и вы их также разрабатываете.

API определяет функциональность, которую предоставляет программа (модуль, библиотека), при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. К тому же, в случае с сайтами мы можем не задумываться о том, какой HTML-код они содержат. Ведь мы будем работать с его другой стороной:



Преимущества API перед основным интерфейсом:

- сокращённый код;
- выше скорость работы программы;
- всё в одном месте;
- не зависит от изменений интерфейса.

Недостатки:

- требует изучения документации.

Формат данных JSON

В контексте сайтов и веб-приложений будем говорить об API как об интерфейсе, который работает с объектами в формате JSON.

JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на языке JavaScript. Несмотря на происхождение от подмножества языкового стандарта ECMA-262 1999 года (к которому и относится JavaScript), формат считается независимым от языка и может использоваться практически с любым языком программирования.

Выглядит JSON так:


```
{
  "coord": {
    "lon": -0.13,
    "lat": 51.51
  },
  "weather": [
    {
      "id": 300,
      "main": "Drizzle",
      "description": "light intensity drizzle",
      "icon": "09d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 280.32,
    "pressure": 1012,
    "humidity": 81,
    "temp_min": 279.15,
    "temp_max": 281.15
  },
  "visibility": 10000,
  "wind": {
    "speed": 4.1,
    "deg": 80
  },
  "clouds": {
    "all": 90
  },
  "dt": 1485789600,
  "sys": {
    "type": 1,
    "id": 5091,
    "message": 0.0103,
    "country": "GB",
    "sunrise": 1485762037,
    "sunset": 1485794875
  },
  "id": 2643743,
  "name": "London",
  "cod": 200
}
```


Формат представления очень похож на словари в языке Python. Доступ к любому элементу в объекте JSON осуществляется так же, как и в случае с коллекциями (последовательностями) в Python — через [] с указанием ключа:

```
print(data['main'][ 'temp'])  # Получаем доступ к температуре
```

Пример работы с API

Рассмотрим в качестве примера запрос погоды с информационного сайта openweathermap.com:

```
import requests
appid = 'b6907d289e10d714a6e88b30761fae22'
service = 'https://samples.openweathermap.org/data/2.5/weather'
req = requests.get(f'{service}?q=London,uk&appid={appid}')
print(req.text)
```

На выходе мы получим текст (в буквальном смысле этого слова: данные, выводимые на экран, в данном случае будут относиться к классу **str**):

```
{"coord":{"lon":-0.13,"lat":51.51},"weather":[{"id":300,"main":"Drizzle","descri
ption":"light intensity
drizzle","icon":"09d"}],"base":"stations","main":{"temp":280.32,"pressure":1012,
"humidity":81,"temp_min":279.15,"temp_max":281.15},"visibility":10000,"wind":{"s
peed":4.1,"deg":80},"clouds":{"all":90},"dt":1485789600,"sys":{"type":1,"id":509
1,"message":0.0103,"country":"GB","sunrise":1485762037,"sunset":1485794875},"id"
:2643743,"name":"London","cod":200}
```

Но нам интересен вывод не как текст, а как объект, словарь, из которого мы получим необходимые данные. Для этих целей существует библиотека JSON (входит в стандартную поставку Python, отдельно устанавливать не нужно) и метод **loads()**, который преобразует переданный ему текст в словарь (dict):

```
import requests
import json
appid = 'b6907d289e10d714a6e88b30761fae22'
service = 'https://samples.openweathermap.org/data/2.5/weather'
req = requests.get(f'{service}?q=London,uk&appid={appid}')
data = json.loads(req.text)
```

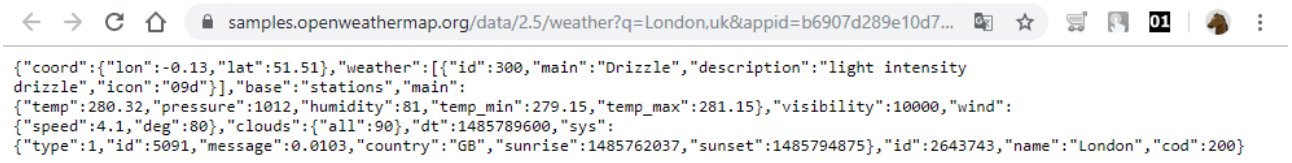
Теперь, имея на руках словарь, можем получить доступ к любым данным. Получим из нашего ответа температуру и название города и выведем их на экран в форматированном виде:

```
import requests
import json
appid = 'b6907d289e10d714a6e88b30761fae22'
service = 'https://samples.openweathermap.org/data/2.5/weather'
req = requests.get(f'{service}?q=London,uk&appid={appid}')
data = json.loads(req.text)
print(f"В городе {data['name']} {data['main']['temp']} градусов по Кельвину")
```

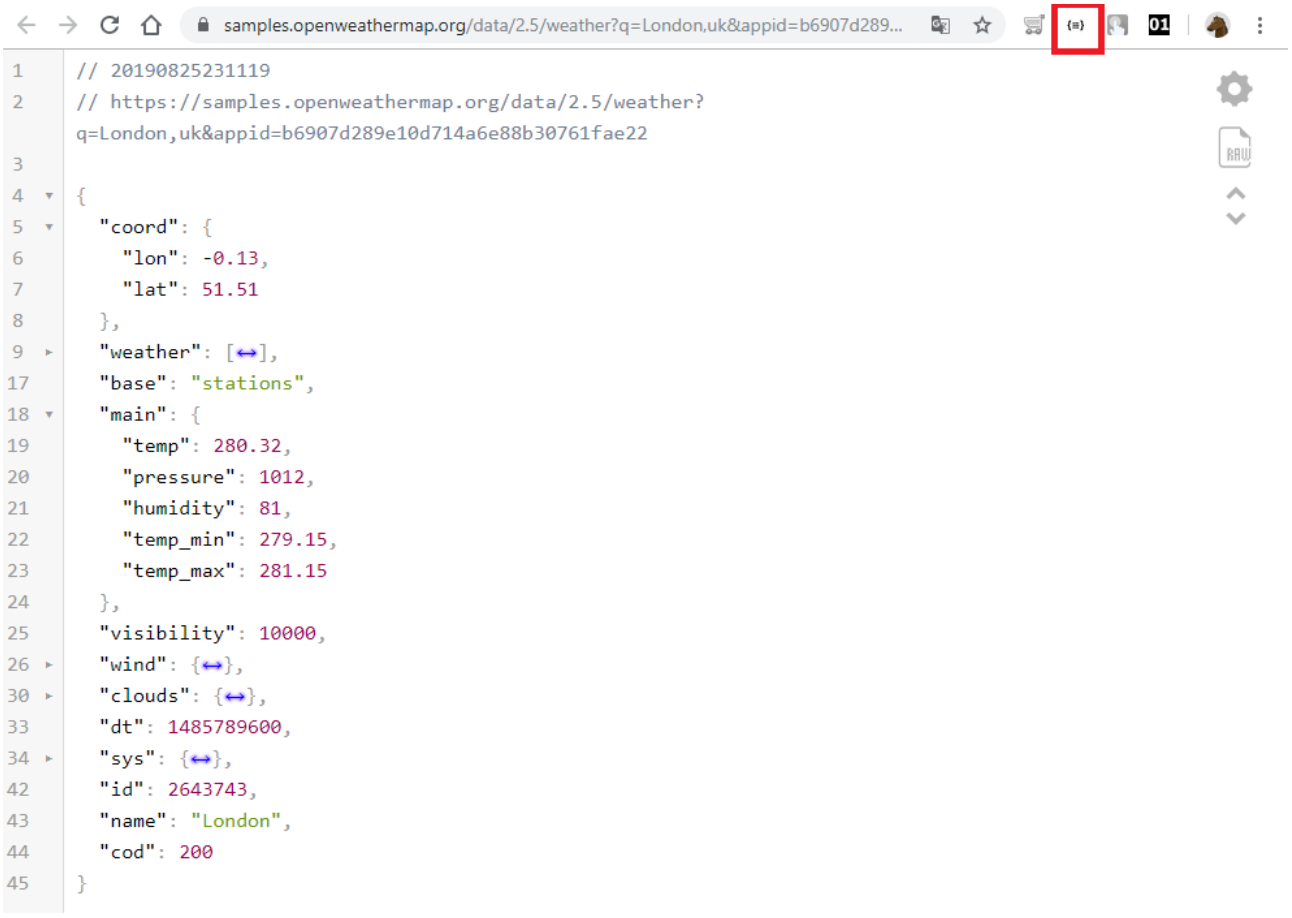
```
В городе London 280.32 градусов по Кельвину
```

Для более удобного изучения содержимого ответа в формате JSON удобно выполнить запрос к API через браузер. Для этого можно просто перейти по ссылке, на которую производится GET-запрос из вашего приложения:

Читать такую выдачу трудно. Но для любого браузера существует расширение **JSON Viewer**, которое придает сплошному тексту удобочитаемый



вид:



Проанализировав данные в таком виде, можно уже легко писать код, получая доступ к информации на любой глубине. Также предварительно можно делать запросы к API через Postman, который тоже выводит информацию в удобном виде.

Глоссарий

Парсинг — это синтаксический анализ информации. Под парсингом HTML, как правило, подразумевают выборочное извлечение большого количества информации с других сайтов и ее последующее использование.

Скрапинг — это сбор данных с различных интернет-ресурсов. Общий принцип его работы можно объяснить следующим образом: автоматизированный код выполняет GET-запросы на целевой сайт и, получая ответ, парсит HTML-документ, ищет данные и преобразует их в заданный формат.

Краулинг — процесс обнаружения и сбора поисковым роботом (краулером) новых и обновлённых страниц для добавления в индекс поисковых систем.

HyperText Transfer Protocol — протокол, созданный для передачи гипертекстовых документов.

HTTPS — это расширение протокола HTTP с поддержкой шифрования для повышения безопасности.

Код состояния HTTP — часть первой строки ответа сервера при запросах по протоколу HTTP. Он представляет собой целое число из трёх десятичных цифр.

HTTP-метод — это указание на основную операцию над ресурсом, то есть команда для сервера. Обычно это короткое слово, написанное заглавными буквами.

Заголовки HTTP — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару имя-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA.

cURL — кроссплатформенная служебная программа командной строки, позволяющая взаимодействовать со множеством различных серверов по множеству различных протоколов с синтаксисом URL.

Postman — позволяет создавать коллекции с запросами к API или веб-сервису. С помощью него можно формировать запросы гораздо более удобным способом, чем cURL, Postman также имеет графический интерфейс.

API — это набор готовых классов, библиотек и программных интерфейсов, которые облегчают программисту написание кода.

JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на языке JavaScript.

Дополнительные материалы

1. Утилита cURL.
2. Утилита Postman.
3. Сервис с хорошей документацией, позволяющий отработать любые типы запросов на простых примерах (также включает примеры с cURL).
4. Списки открытых API.
5. Р. Митчелл. «Скрапинг веб-сайтов с помощью Python».
6. Seppe van den Broucke, Bart Baesens. Practical Web Scraping for Data Science: Best Practices and Examples with Python.
7. Вандер Плас Дж. «Python для сложных задач. Наука о данных и машинное обучение».
8. Грас Джоэл. «Data Science. Наука о данных с нуля».
9. Силен Д., Мейсман А. «Основы Data Science и Big Data. Python и наука о данных».

Домашнее задание

- Посмотреть документацию к API GitHub, разобраться как вывести список репозиториев для конкретного пользователя, сохранить JSON-вывод в файле *.json.
- Изучить список открытых API. Найти среди них любое, требующее авторизацию (любого типа). Выполнить запросы к нему, пройдя авторизацию. Ответ сервера записать в файл.

Настоятельно рекомендуем сдавать практическое задание в виде ссылки на pull request.

Рекомендуемый способ организации данных в репозитории: создать отдельные папки по темам и помещать в них отдельные файлы с правильным расширением для каждой задачи.

Используемая литература

1. Протокол HTTP.
2. Коды состояния.
3. Информация по кодам заголовков.
4. О HTTP-авторизации.
5. Документация по cURL.
6. Официальная документация модуля Kequests.
7. Описание формата данных JSON.
8. Документация по модулю JSON в Python.
9. Документация по API OpenWeatherMap.