



Методы сбора и обработки данных при помощи Python

Основы клиент-серверного взаимодействия. Парсинг API

**На этом уроке**

1. Узнаем основные принципы сбора данных.
2. Поймём, как отправлять GET-запросы с помощью разных инструментов.
3. Разберёмся, как работать с ответами от сервера и API, JSON

**Оглавление**

[Парсинг, скрапинг, краулинг](#_4d34og8)

[Парсинг](#_2s8eyo1)

[Скрапинг](#_17dp8vu)

[Краулинг](#_26in1rg)

[Robots.txt](#_ro5pzy9csqvc)

[HTTP и HTTPS](#_lnxbz9)

[HTTP-коды](#_35nkun2)

[HTTP-методы](#_1ksv4uv)

[HTTP-заголовки](#_44sinio)

[GET-запросы](#_2jxsxqh)

[GET-запросы с помощью cURL](#_z337ya)

[GET-запросы с помощью Postman](#_3j2qqm3)

[GET-запросы в Python](#_1y810tw)

[API](#_4i7ojhp)

[Что такое API](#_2xcytpi)

[Формат данных JSON](#_1ci93xb)

[Пример работы с API](#_3whwml4)

[Глоссарий](#_2bn6wsx)

[Дополнительные материалы](#_qsh70q)

[Используемая литература](#_3as4poj)

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

*Вам предстоит работать с командной строкой. Если вы используете ОС Windows, рекомендуем 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, важно своевременно получать актуальные данные о движении рынка (динамике цен и товаров) и локальные новости, которые часто влияют на формирование спроса.

Сегодня объёмы информации превосходят возможности обработать их у любого, даже самого талантливого узкопрофильного специалиста. Поэтому для автоматического сбора и обработки больших объёмов информации придумали методы, знание и умелое использование которых — одно из важнейших свойств новой профессии — Data Scientist.

## Парсинг

Парсинг — это синтаксический анализ информации. Под парсингом подразумевают поиск и выборочное извлечение определённой информации с разных сайтов при помощи HTML-разметки. Если вы ещё не знакомы с HTML-разметкой, не волнуйтесь, мы будем подробнее говорить о ней в следующих уроках.

## Скрапинг

В широком понимании веб-скрапинг — это сбор данных с различных интернет-ресурсов. Общий принцип его работы можно объяснить так: автоматизированный код выполняет GET-запросы на целевой сайт и, получая ответ, парсит HTML или JSON ответ, ищет данные и преобразует их в заданный формат.

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

* отдельные сервисы, которые работают через API или имеют веб-интерфейс (Embedly, DiffBot и другие);
* проекты на разных языках программирования с открытым кодом (для Python это, например, Scrapy, с которым мы подробнее познакомимся на следующих уроках).

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

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

1. HTML на разных сайтах почти всегда разная. Кроме этого, даже на одном сайте разметка может меняться со временем, так что, если вы написали парсер, основанный на HTML-разметке, будьте готовы к тому, что в одни прекрасный момент он может перестать работать.
2. Масса веб-ресурсов использует HTML5, где каждый элемент может быть абсолютно уникальным. Однако в таком случае можно использовать не DOM-селекторы, а структуру и вложенность элементов.
3. Некоторые ресурсы защищены от копирования данных (многоуровневая вёрстка, использование JavaScript для рендеринга контента, проверка user agent и так далее), а значит и от скрапинга. Существуют методы обхода этих ограничений, однако не стоит забывать об этичном парсинге. Старайтесь не нарушать права владельцев и не ломать сайты. Об этичном парсинге чуть ниже.

## Краулинг

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

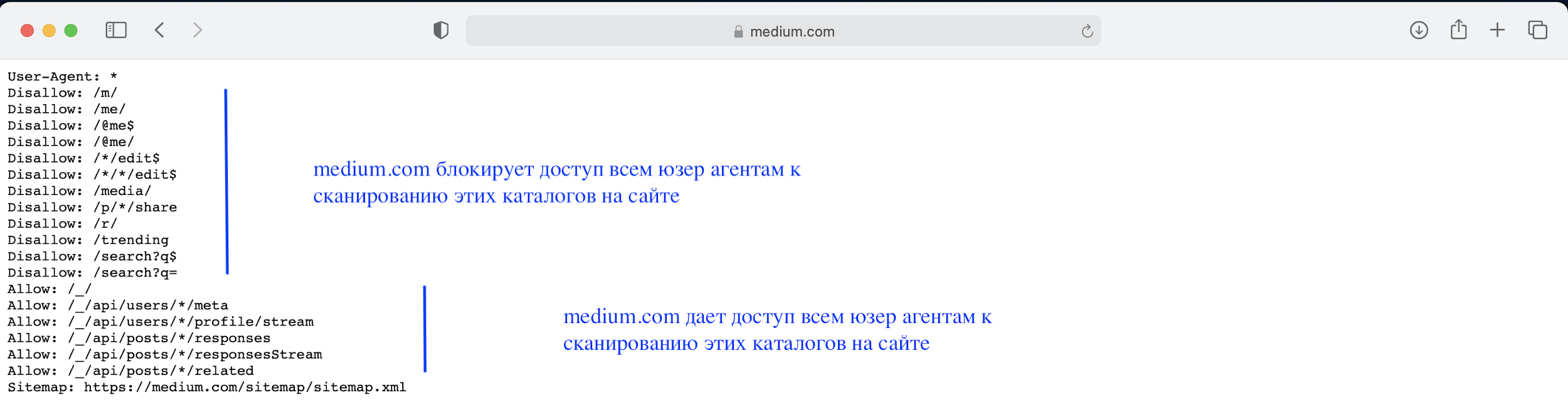
Поисковый робот (он же crawler, краулер, паук, бот) — программа для сбора контента в интернете. Краулер состоит из множества компьютеров, запрашивающих и выбирающих страницы намного быстрее, чем пользователь с помощью веб-браузера. Фактически он может запрашивать тысячи разных страниц одновременно.

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.

### Robots.txt

Это файл, которым руководствуются краулеры при индексации сайта. В robots.txt указано, что и кому можно парсить.

Вот, например, файл robots.txt с сайта Medium.com:



При парсинге сайта с помощью requests этот файл не учитывается. Однако вам следует ознакомиться с ним перед тем, как начинать парсить новый сайт. Возможно, владелец не ограничил доступ ни к каким материалам. При использовании же Scrapy для парсинга вам надо будет указать, принимать во внимание robots.txt сайта или нет (вернёмся к этому в уроке по Scrapy).

Этим же файлом стоит руководствоваться и вам при написании парсера. А также помнить о том, что парсинг должен быть этичным. Вот основные принципы этичного парсинга:

* **API — часто лучшее решение.** У некоторых веб-сайтов есть собственные API, созданные специально для вас, чтобы получать данные без парсинга. Это означает две вещи: вы будете делать это по правилам сайта и у вас есть полномочия на получение информации. Если есть API, используйте его вместо парсинга.
* **Уважайте robots.txt.** Файл robots.txt, также известный как стандарт исключения роботов, указывает на программное обеспечение для веб-сканирования, где это разрешено (или запрещено) на веб-сайте. Это часть протокола исключения роботов (REP), который представляет собой группу веб-стандартов, созданных для регулирования того, как роботы сканируют интернет.
* **Прочитайте условия использования.** Это основной источник, в котором владелец сайта сообщает вам правила нахождения и использования размещённой информации. Да, проще просто нажать «Я согласен» или «Принимаю» и надеяться на лучшее. Помните, что владельцы сайта создали эти правила это не просто так.
* **Будьте аккуратны.** Процесс парсинга может быть довольно жёстким для серверов, а агрессивный парсинг иногда может приводить к проблемам с функциональностью, создавая неудобства для пользователей. Возьмите за привычку выполнять парсинг в непиковые часы посещения сайта, например, ночью. И не забудьте делать паузы между запросами, чтобы владелец сайта не спутал ваш парсинг с DDoS-атакой.
* **Цените содержание, которое вы храните.** Вы должны брать только тот контент, который вам нужен. У вас также всегда должна быть достаточно веская причина для парсинга данных с сайта. Цель использования данных — создать больше ценности, а не просто дублировать информацию.
* **Относитесь к данным с уважением.** Вам дали разрешение на использование контента, но это не значит, что теперь вы можете предоставлять его другим пользователям. Не выдавайте эту информацию за собственную.
* **Возвращайте, когда можете.** Постарайтесь отблагодарить владельцев сайта за предоставленные данные. Например, вы можете упомянуть сайт в статье, которую создали на основе информации, полученной с него. Таким образом, вы сможете повысить трафик реальных пользователей на сайт.
* **Практикуйте этич ный парсинг сайтов.** Потребность в источниках данных со временем возрастает, тогда как на многих веб-сайтах нет собственных API, с помощью которых разработчики могли бы получить доступ к нужным данным. Это означает, что парсинг данных со временем будет возрастать, поэтому разработчикам важно знать, как это делать правильно.

Теперь поговорим о протоколах и кодах.

# 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 повышенной 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. 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.

Выделено пять классов кодов состояния:

| **1xx: Information** | Коды информируют о процессе передачи. Обычно это предварительный ответ, который состоит только из Status Line и опциональных заголовков, а завершается пустой строкой. Обязательных заголовков для этого класса нет. |
| --- | --- |
| **2xx: Success** | Коды указывают, что запрос клиента успешно получен, понят, и принят. Примеры ответов — 200 OK, 201 Created. |
| **3xx: Redirect** | Коды показывают, 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. |
| **4xx: Client Error** | Класс кодов предназначен для определения ошибок, которые произошли на стороне клиента. К примеру, ошибка 404 Not Found возникает, когда клиент обращается к ресурсу, которого не существует. |
| **5xx: Server Error** | Коды ответов 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. Например, ошибка 500 Internal Server Error возникает, когда сервер недоступен. |

## 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 создаёт его и возвращает статус 201 (Created). Если 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 специфичны и применяются редко.

## HTTP-заголовки

Заголовки HTTP — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару имя-значение.

Все заголовки разделяются на четыре основные группы:

1. **General Headers** должны включаться в любое сообщение клиента и сервера. Пример: *User-Agent*, который указывает программное обеспечение клиента и его характеристики.
2. **Request Headers** используются только в запросах клиента. Пример: *Authorization*, в котором передаются данные для авторизации.
3. **Response Headers** — только для ответов от сервера. Пример: *Age*, который хранит количество секунд с момента модификации ресурса.
4. **Entity Headers** — сопровождают каждую сущность сообщения, используются при передаче множественного содержимого. 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, то есть 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, возвращённые сервером в файл;
* **-e** — задать поле Referer URI, которое указывает, с какого сайта пришёл 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/">hereОК</A>.  </BODY></HTML>  \* Connection #0 to host google.com left intact |
| --- |

Вывод содержит несколько блоков важной информации:

* строки, начинающиеся с символа **\*** — **системные сообщения**, генерируемые ОС;
* строки, начинающиеся с символа **>** — **строки запроса** к серверу;
* строки, начинающиеся с символа **<** — **строки ответа** от сервера;
* Строка GET / HTTP/1.1 — **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.

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 защищённому 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.oAHGFs4OQUcoHRYmC4QsETXsDISEeJUByNhQHdNLzqU; Path=/; HttpOnly  < WWW-Authenticate: Basic realm="Users"  < Content-Length: 12  < Connection: keep-alive |
| --- |

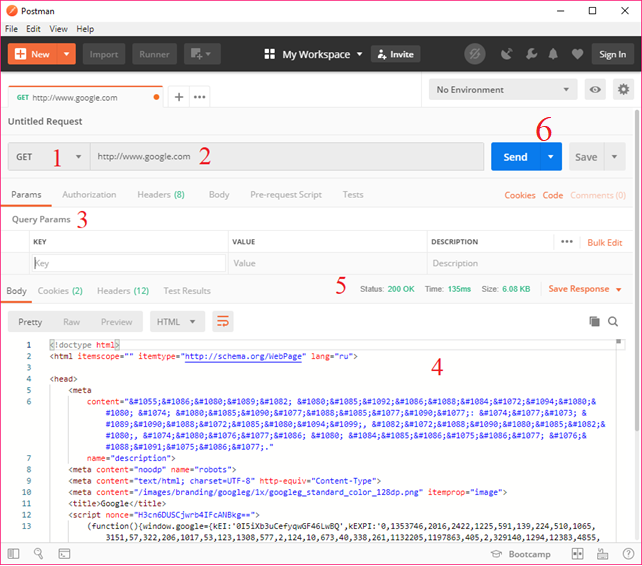
Мы видим HTTP-код 401 — статус ответа, который говорит нам, что требуется авторизация. Также мы видим заголовок WWW-Authenticate, который начинается со слов Basic realm, — значит, применена BA. Если выполнить такой запрос в браузере, то браузер, получив подобный ответ от сервера, покажет окошко для ввода логина и пароля.

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

| curl --help |
| --- |

## GET-запросы с помощью Postman

Основное предназначение приложения Postman — создание запросов 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, можно выбрать любой тип запроса;

2 — адрес сервиса, к которому отправляется запрос;

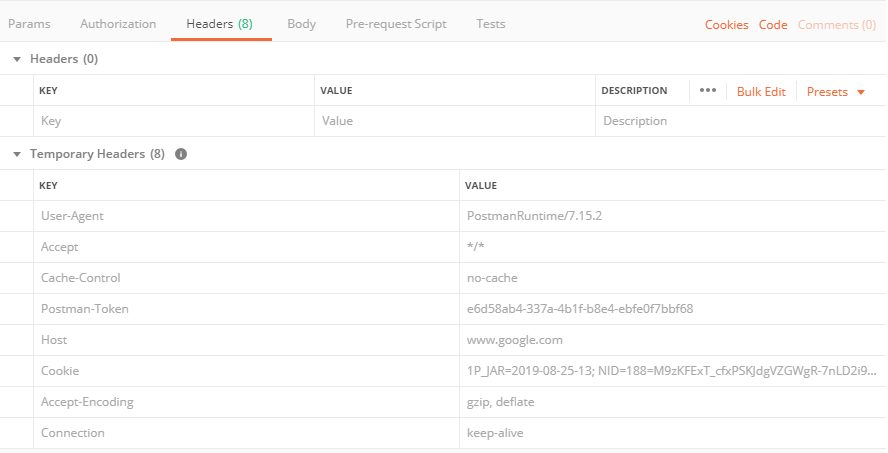
3 — параметры запроса;

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.

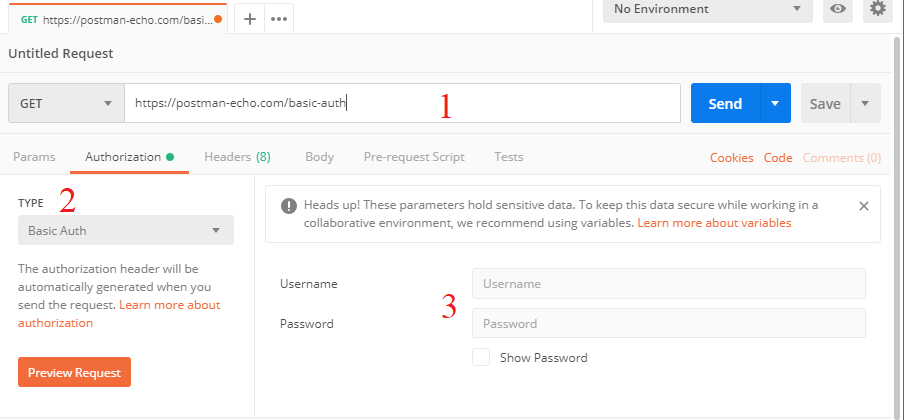
**Важно!** *Чтобы избежать дополнительных проблем при запросах, лучше передавать вместе с вашим запросом заголовок, имитирующий запрос из реального браузера. Для этого передайте заголовок User-Agent:*

*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.*

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



На рисунке:

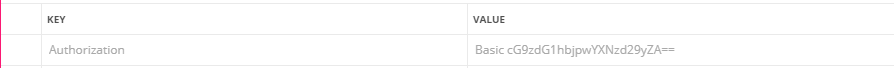
1 — сервис, к которому отправляем запрос;

2 — выпадающий список с разными типами авторизации;

3 — поля для ввода логина и пароля.

Вводим в поле Key «Authorization», а в поле Value - «Basic cG9zdG1hbjpwYXNzd29yZA==».

При нажатии на кнопку Send будет 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 позволяет легко выполнять любые 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.net--></body></html> |
| --- |

Как и в случае с cURL, заголовки здесь необходимо формировать вручную. 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.json()) |
| --- |

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.w2TqQA5TwkFiz475tyvCBrvrjstjmFCMN3kY1IP9LL8; 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. специальный протокол для взаимодействия компьютерных программ, который позволяет использовать функции одного приложения внутри другого.

Например, на сайте «Главред» есть сервис для улучшения текстов. Чтобы воспользоваться сервисом и проанализировать свой текст, пользователям нужно было заходить на сайт. А потом разработчики «Главреда» добавили API. Теперь разработчики других платформ могут встроить сервис «Главреда» к себе, чтобы пользователи могли проводить анализ текста, не покидая приложение и не переходя на другой сайт.

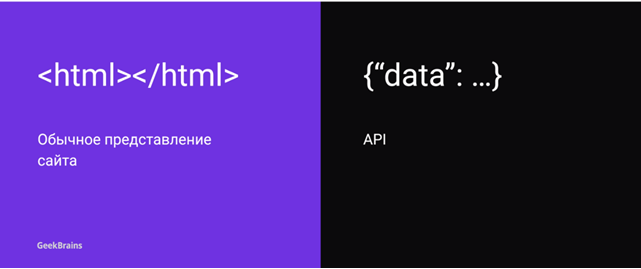
Или другой пример — быстрая регистрация с помощью аккаунта в соцсетях. Приложение может использовать API социальной сети, чтобы предоставить пользователю упрощённый доступ.

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 сервисы (например. Twitter или Facebook), позволяющие получить данные программным путём. 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, передаёт 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.

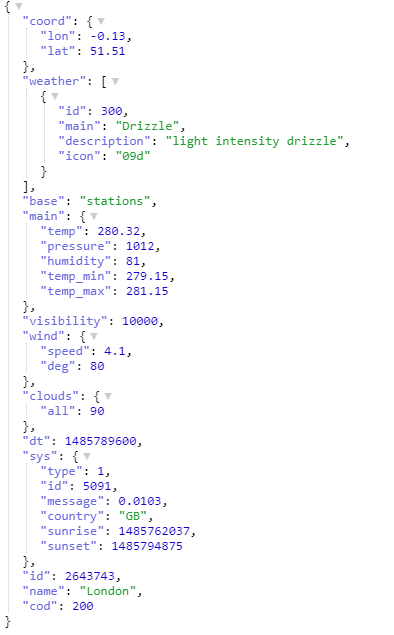
Всегда смотрите, нет ли у сайта, который вы собираетесь парсить, API. Вдруг вам повезло, и вы сможете получить все данные, выполнив один простой запрос.

## 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. Формат считается независимым от языка и может использоваться практически с любым языком программирования.

Выглядит JSON так:



Формат представления очень похож на словари в языке Python. Доступ к любому элементу в объекте JSON осуществляется так же, как и в случае со словарями 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' url = 'https://samples.openweathermap.org/data/2.5/weather' req = requests.get(f'{url}?q=London,uk&appid={appid}') print(req.json()) |
| --- |

На выходе мы получим текст json-объект:

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

Теперь, получив ответ, можем получить доступ к любым данным.

Мы можем немного по-другому сформировать запрос, используя параметры. Как вы видите, после ссылки идёт вопросительный знак, который означает, что всё, что написано дальше, — это параметры обращения к сайту. Параметры разделены знаком &, всё, что находится слева от знака = это ключ, всё, что справа — значение.

В данном примере параметров не так много, поэтому мы можем использовать их в ссылке без создания словаря. Но бывает, что параметров очень много или какие-то из них должны меняться, например, номер страниц при парсинге многостраничного сайта. В таком случае удобнее использовать словарь. Создадим словарь params и запишем туда наши параметры.

| params = {  'q': 'London,uk',  'appid': 'b6907d289e10d714a6e88b30761fae22'  } |
| --- |

И теперь сформируем запрос, используя этот словарь, и проверим статус:

| url = '<https://samples.openweathermap.org/data/2.5/weather>'  r = requests.get(url=url, params=params)  print(r.status\_code) |
| --- |

В get запросе requests вы также можете при необходимости указать заголовки и куки, предварительно создав словари headers и cookies соответственно.

| r = requests.get(url=url, headers=headers, cookies=cookies, params=params) |
| --- |

Затем получим из нашего ответа температуру и название города и выведем их на экран в форматированном виде:

| data = r.json()  print(f"В городе {data['name']} {data['main']['temp']} градусов по Кельвину") |
| --- |

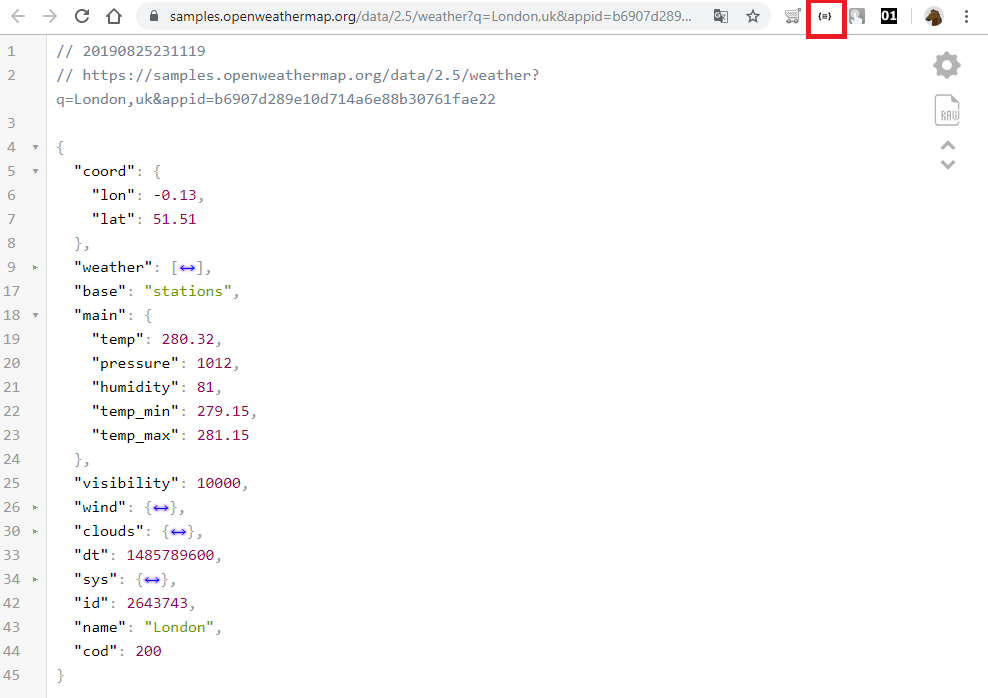
| В городе London 280.32 градусов по Кельвину |
| --- |

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

Чтобы получить ссылку полностью, можно написать

| req.url |
| --- |

Полученную ссылку открываем через Firefox.



Проанализировав данные в таком виде, можно уже легко писать код, получая доступ к информации на любой глубине. Тут всё подсвечено и видна вложенность ключей друг в друга.

# Глоссарий

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

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

**Заголовки HTTP** — 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** — 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](https://curl.haxx.se/download.html).
2. [Утилита Postman](https://www.getpostman.com/products).
3. [Сервис с хорошей документацией, позволяющий отработать любые типы запросов на простых примерах (также включает примеры с cURL).](https://postman-echo.com)
4. [Списки открытых API](https://www.programmableweb.com/apis/directory).
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».

# Используемая литература

1. [Протокол HTTP](https://ru.wikipedia.org/wiki/HTTP).
2. [Коды состояния](https://ru.wikipedia.org/wiki/%252525D0%252525A1%252525D0%252525BF%252525D0%252525B8%252525D1%25252581%252525D0%252525BE%252525D0%252525BA_%252525D0%252525BA%252525D0%252525BE%252525D0%252525B4%252525D0%252525BE%252525D0%252525B2_%252525D1%25252581%252525D0%252525BE%252525D1%25252581%252525D1%25252582%252525D0%252525BE%252525D1%2525258F%252525D0%252525BD%252525D0%252525B8%252525D1%2525258F_HTTP).
3. [Информация по кодам заголовков](https://ru.wikipedia.org/wiki/%252525D0%252525A1%252525D0%252525BF%252525D0%252525B8%252525D1%25252581%252525D0%252525BE%252525D0%252525BA_%252525D0%252525B7%252525D0%252525B0%252525D0%252525B3%252525D0%252525BE%252525D0%252525BB%252525D0%252525BE%252525D0%252525B2%252525D0%252525BA%252525D0%252525BE%252525D0%252525B2_HTTP).
4. [О HTTP-авторизации.](https://developer.mozilla.org/ru/docs/Web/HTTP/%252525D0%25252590%252525D0%252525B2%252525D1%25252582%252525D0%252525BE%252525D1%25252580%252525D0%252525B8%252525D0%252525B7%252525D0%252525B0%252525D1%25252586%252525D0%252525B8%252525D1%2525258F)
5. [Документация по cURL](https://curl.haxx.se/docs/manual.html).
6. [Официальная документация модуля Кequests](https://2.python-requests.org/en/master/).
7. [Описание формата данных JSON](https://ru.wikipedia.org/wiki/JSON).
8. [Документация requests](https://docs.python-requests.org/en/latest/user/quickstart/).
9. [Документация по модулю JSON в Python](https://docs.python.org/3/library/json.html).
10. [Документация по API OpenWeatherMap](https://openweathermap.org/api).