

# Работа с библиотекой requests, http запросы

Тимур Анвардинов  
Инженер по контролю качества, [smotreshka.tv](https://smotreshka.tv)

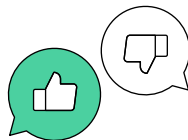


# Проверка связи





## Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



## Поставьте в чат:

-  если меня видно и слышно
-  если нет

# Рекомендации

## → Если смотрите с компьютера

- Используйте браузеры **Google Chrome** или **Microsoft Edge**
- Если есть проблемы с изображением или звуком, обновите страницу — **F5**

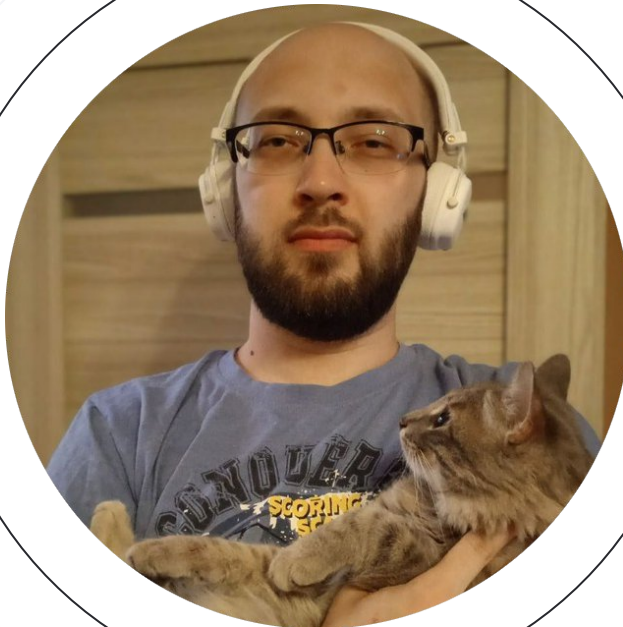
## → Если смотрите с мобильного телефона или планшета

- Перейдите с мобильного интернет-соединения на **Wi-Fi**
- Если есть проблемы с изображением или звуком, перезапустите приложение **МТС Линк** на телефоне
- Предварительно проверьте, подходит ли ваше устройство для подключения к вебинару, по [ссылке](#)

# Тимур Анвартдинов

## О спикере:

- Организовал с нуля автоматизацию тестирования в своей компании
- Отвечаю за процесс обучения новых сотрудников
- Вырос из студентов Нетологии в преподаватели



# Сегодня вы узнаете

- 1 Что такое HTTP
- 2 Как выглядит клиент-сервисное взаимодействие
- 3 Как работает HTTP запрос
- 4 Синтаксис и коды HTTP
- 5 Как организована структура url
- 6 Какие могут быть параметры в адресе
- 7 Для чего нужны заголовки, а для чего - параметры?
- 8 Что такое библиотека Requests
- 9 Методы выполнения запросов



# План занятия

- 1 Что такое HTTP
- 2 Client-Server
- 3 HTTP запрос
- 4 HTTP методы
- 5 URI и URL
- 6 Заголовки и параметры
- 7 Ответ сервера
- 8 Библиотека requests



# Что такое HTTP



1

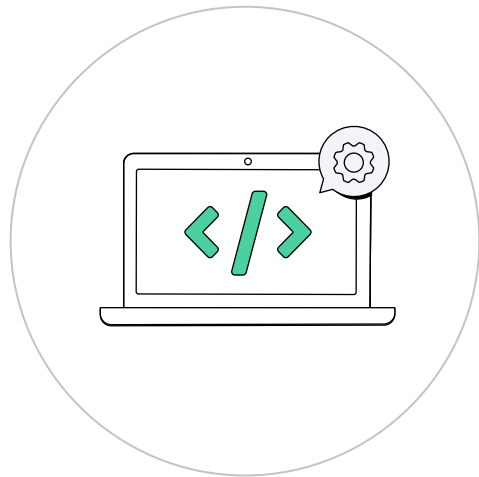
# HTTP

**HTTP** (HyperText Transfer Protocol) — это протокол, по которому ваш компьютер (или программа) общается с веб-сервером. Он основан на модели «клиент-сервер»:

**Клиент** — это браузер, мобильное приложение или, например, Python-скрипт с библиотекой requests.

**Сервер** — это удалённая машина, которая обрабатывает запросы и отправляет ответы.

Пример: вы заходите на сайт. Браузер делает HTTP-запрос, сервер присылает HTML-страницу — и вы её видите.





# Основные свойства протокола HTTP

1

## Клиент-серверная модель

HTTP построен на архитектуре клиент-сервер.

2

## Протокол прикладного уровня

Он не заботится о транспортировке данных, а фокусируется на структуре и содержанием передаваемой информации

3

## Без состояния (stateless)

Каждое соединение независимо от предыдущего. Сервер не запоминает, что было до этого запроса (если специально не использовать куки или токены).

4

## Расширяемость

Можно добавлять собственные заголовки, методы и другие расширения.



# Основные свойства протокола HTTP

## 5 Человеко-читаемый текстовый протокол

Запросы и ответы — обычный текст, легко читаются и отлаживаются. Это делает HTTP удобным для изучения и диагностики.

## 6 Поддержка разных типов данных

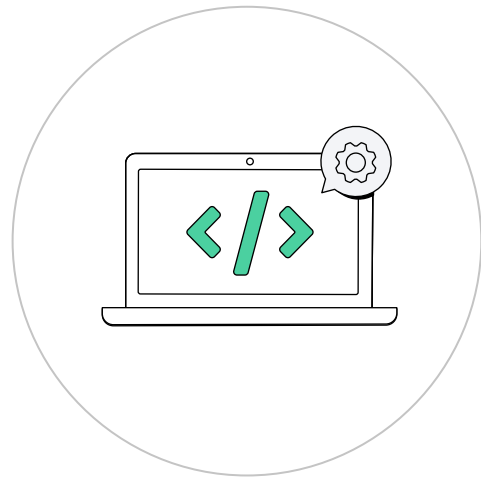
HTTP может передавать любой тип данных — HTML, JSON, XML, изображения, видео и т.д. Определяется через заголовок Content-Type.

## 7 Поддержка кэширования

Сервер может указывать, можно ли кэшировать ответ (Cache-Control, ETag и др.). Это повышает производительность и снижает нагрузку.

## 8 Поддержка авторизации

HTTP поддерживает заголовки для передачи авторизации. Это позволяет защищать API и веб-страницы.



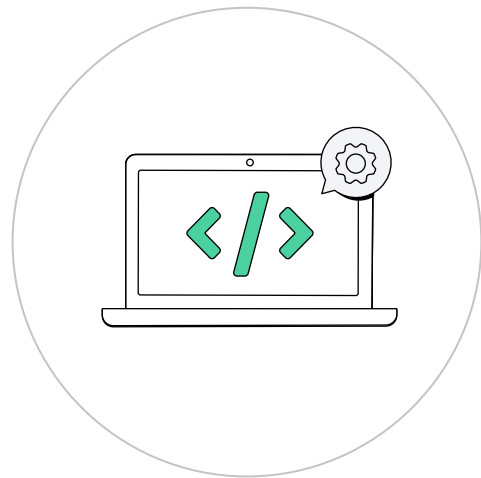
# Основные свойства протокола HTTP

## 9 Использование URI (Uniform Resource Identifier)

Каждому ресурсу соответствует уникальный адрес (URI).

## 10 Возможность работы по защищённому соединению (HTTPS)

HTTP + шифрование через SSL/TLS = HTTPS. Обеспечивает конфиденциальность и целостность данных



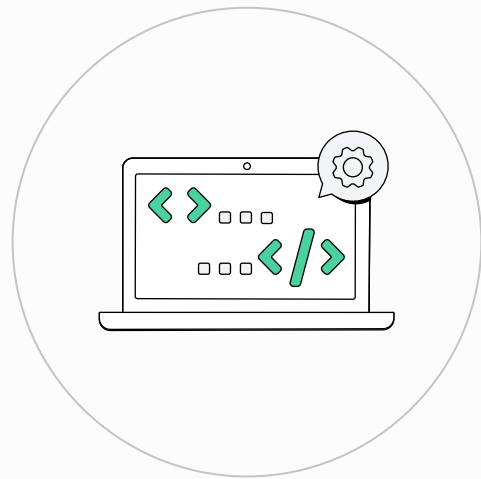
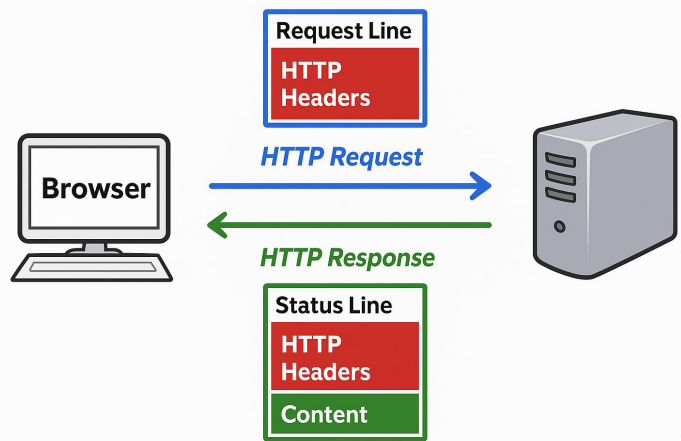
# Client-Server



2

# Клиент-серверное взаимодействие

В роли клиента может выступать не только браузер, но и любое устройство и даже другая программа. Когда говорят «клиент-сервер», подразумевают, что есть сторона, которая запрашивает данные, а другая сторона ей отвечает.

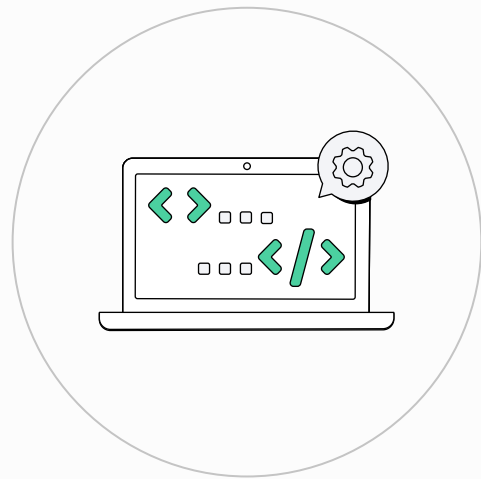


# Так с чем именно взаимодействует клиент?

**API (Application Programming Interface)** — это интерфейс (набор правил), который позволяет одной программе взаимодействовать с другой.

API — это как меню в ресторане. Ты выбираешь из него (делаешь запрос), а кухня (программа или сервер) приносит тебе блюдо (ответ).

Ты не знаешь, как именно готовят на кухне — тебе это не нужно. Главное, что ты знаешь, что можно запросить и как это сделать.

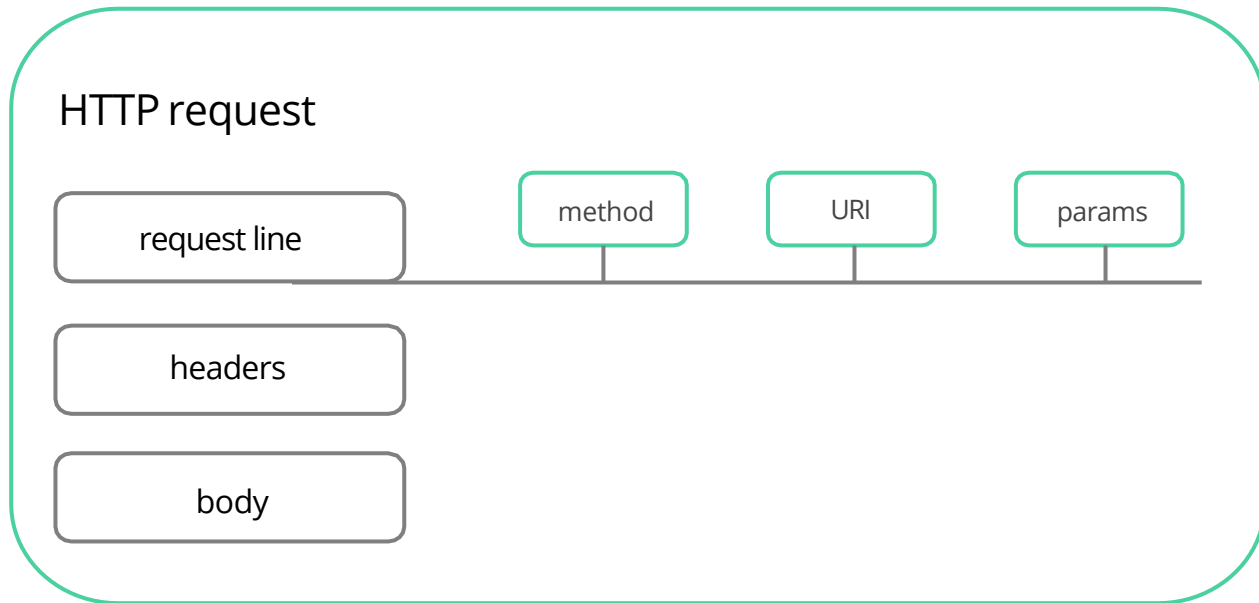


# HTTP запрос

3



# Схема запроса (HTTP Request)





# Структура HTTP-запроса

**HTTP-запрос** состоит из 3-х основных частей (и обязательной пустой строки):

## 1 Стартовая строка запроса (Request Line)

<метод> <путь> <версия>  
GET /articles?id=42 HTTP/1.1

## 2 Заголовки (Headers)

Заголовки описывают, что именно мы отправляем и чего ожидаем.  
Authorization: Bearer <токен>  
Content-Type: application/json

## 3 Пустая строка (разделяет заголовки и тело).

## 4 Тело запроса (Body) — не всегда

Используется в методах типа POST, PUT, PATCH — когда нужно передать данные на сервер. Формат тела определяется заголовком Content-Type. Тело запроса отделяется пустой строкой от блока заголовков.

# Пример полного HTTP-запроса

```
POST /login HTTP/1.1  
Host: example.com  
User-Agent: Python/3.10  
Content-Type: application/json  
Content-Length: 48
```

```
{  
  "username": "admin",  
  "password": "1234"  
}
```

# HTTP методы

4

# HTTP методы

Метод	Назначение	Используется когда
GET	Получить данные	Хочешь получить информацию
POST	Отправить данные (создание)	Отправляешь форму, создаёшь ресурс
PUT	Создать / заменить ресурс	Создаёшь ресурс, полностью меняешь ресурс
PATCH	Частично обновить ресурс	Обновляешь только часть данных
DELETE	Удалить ресурс	Хочешь удалить что-то
HEAD	Как GET, но без тела ответа	Проверяешь, существует ли ресурс
OPTIONS	Возвращает допустимые методы	Узнаёшь, что вообще можно делать с ресурсом

# HTTP методы

## GET

- Самый часто используемый
- Не изменяет данные на сервере (должен быть идемпотентным)
- Можно передавать параметры в URL

## POST

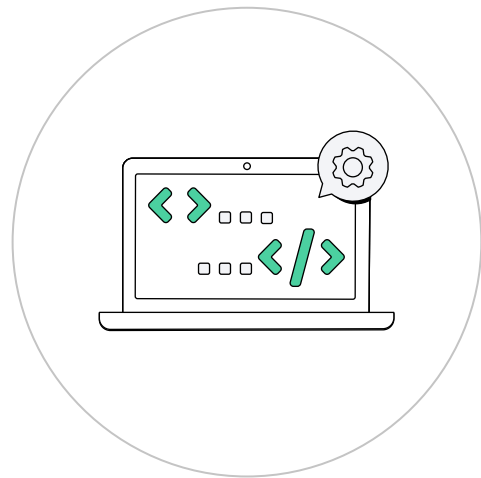
- Используется для отправки данных и создания новых объектов
- Данные отправляются в теле запроса (body)

## PUT

- Создание или полная замена существующего ресурса
- Требуется отправить все поля объекта, даже те, что не изменились
- Данные отправляются в теле запроса (body)

## PATCH

- Частичное обновление объекта
- Используется, если хочешь изменить только одно-два поля





# **Что такое “идемпотентность”?**

Напишите ваш ответ в чат

# HTTP методы

## DELETE

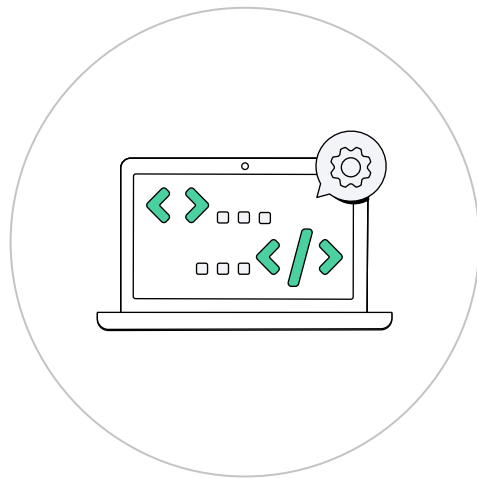
- Удаляет ресурс
- Обычно не требует тела запроса

## HEAD

- Как GET, но без тела в ответе. Используется для проверки заголовков (например, Content-Type, размер и т.п.)

## OPTIONS

- Запрос, который показывает, какие методы разрешены для ресурса

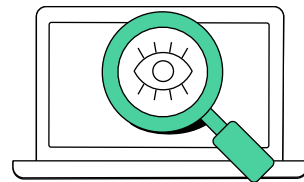


# Полезные свойства методов

Метод	Идемпотентность	Изменяет данные?	Тело запроса?
GET	Да	Нет	Нет
POST	Нет	Да	Да
PUT	Да	Да	Да
PATCH	Нет	Да	Да
DELETE	Да	Да	Нет



# Демонстрация работы



# URI и URL

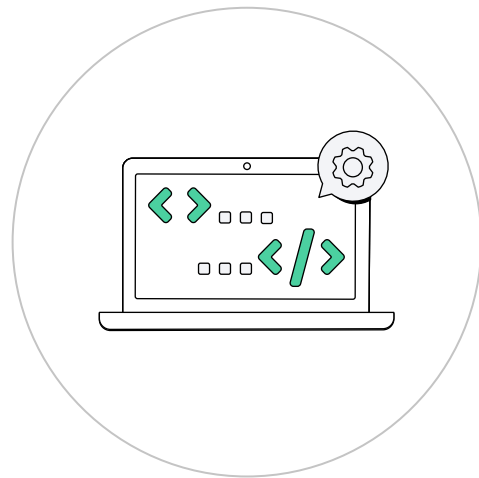


5

# Что такое URI и URL

## URI (Uniform Resource Identifier)

Унифицированный идентификатор ресурса. Он обозначает что-то, к чему можно обратиться в интернете или локальной сети. Это может быть файл, страница, API-метод и т.д.





# Чем отличается URI и URL?

Напишите ваш ответ в чат

# Что такое URI и URL

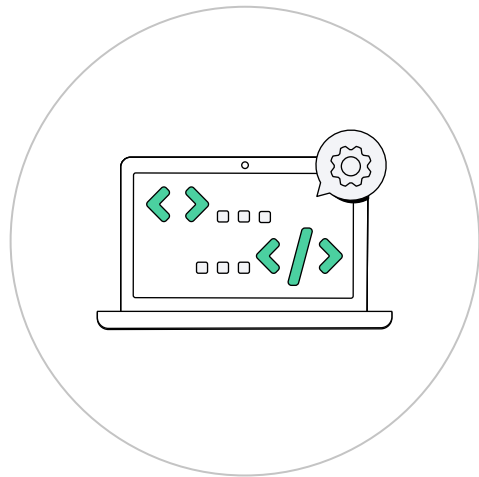
## URI (Uniform Resource Identifier)

Унифицированный идентификатор ресурса. Он обозначает что-то, к чему можно обратиться в интернете или локальной сети. Это может быть файл, страница, API-метод и т.д.

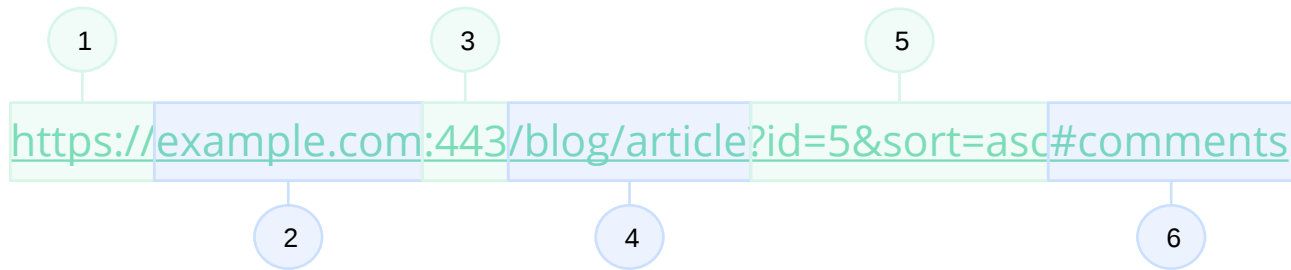
## URL (Uniform Resource Locator)

Подтип URI, который указывает:

- Где находится ресурс (его адрес)
- Как к нему обратиться (протокол: http, https, ftp...)

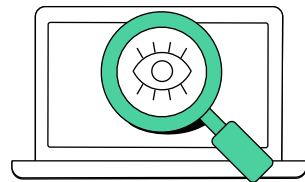


# Структура URL



Часть URL	Пример	Что это означает
1. Схема (протокол)	https	Как обращаться (HTTP, HTTPS, FTP...)
2. Хост (домен)	example.com	Где находится ресурс (адрес сервера)
3. Порт (необяз.)	:443	Через какой порт (по умолчанию 80 или 443)
4. Путь	/blog/article	Какой именно ресурс на сервере
5. Параметры	?id=5&sort=asc	Доп. параметры (query string)
6. Якорь	#comments	Внутренняя ссылка на часть страницы (на клиенте)

# Демонстрация работы



# Заголовки и параметры



6



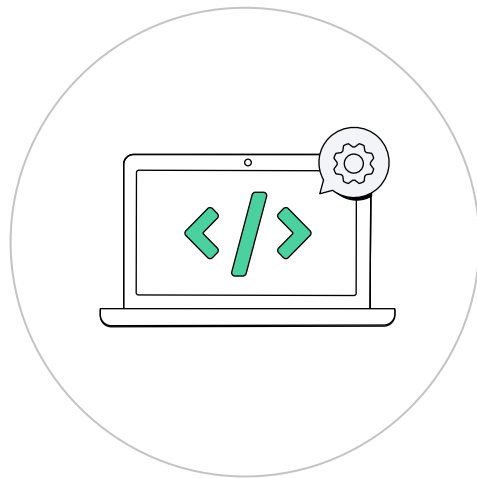
# Что такое HTTP-заголовки (Headers)?

## HTTP-заголовки

Данные в формате **ключ: значение**, отправляемые в запросе или ответе. Они дают метаинформацию: о запросе, клиенте, формате данных, языке, безопасности и т.д.

## Когда использовать заголовки:

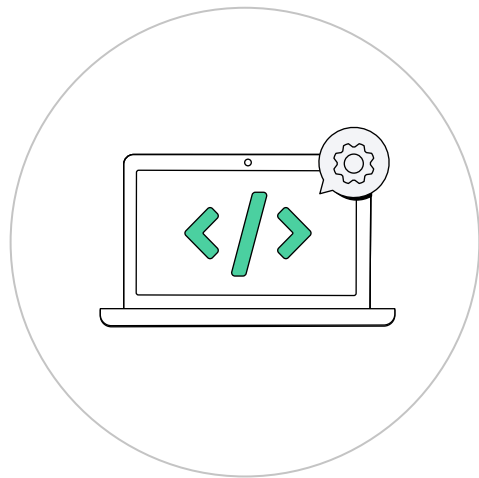
- Авторизация (Authorization: Bearer <токен>)
- Тип отправляемых или ожидаемых данных (Content-Type, Accept)
- Язык и локализация: (Accept-Language: ru-RU)
- Инфо о клиенте: (User-Agent)
- Кэш, cookies, контроль запросов и т.п.



# Что такое query-параметры?

**Query-параметры** добавляются к URL после знака **?** в виде **ключ=значение** и передаются на сервер как часть запроса. Они используются для фильтрации, сортировки и передачи входных данных в GET-запросах (например, рекламных меток).

Можно задать сразу несколько, разделяя пары с помощью знака **&**.



# Для чего нужны заголовки и параметры

## Заголовки

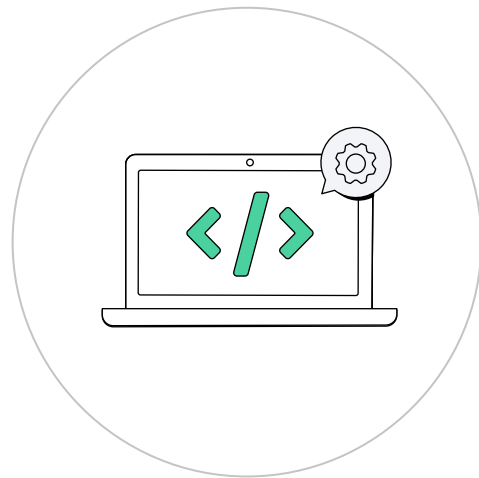
Как правило, передают информацию о запросе. Например, авторизационные данные пользователя, версию браузера, выставленные cookie, поддерживаемые форматы сжатия данных и т.д.

## Параметры

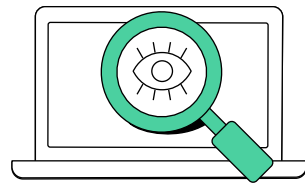
Содержат информацию о том, что именно запрашивает пользователь. Также в параметрах запроса можно передавать произвольную информацию при размещении ссылки.

## Например:

Источник, где размещена ссылка (это используется в интернет-рекламе)



# Демонстрация работы



# Ответ сервера

7

# Структура HTTP ответа

## 1 HTTP-запрос

Состоит из 3-х основных частей (и обязательной пустой строки)

## 2 Стартовая строка (Status Line) (HTTP/1.1 200 OK)

Версия протокола, код статуса и его описание

## 3 Заголовки ответа (Headers)

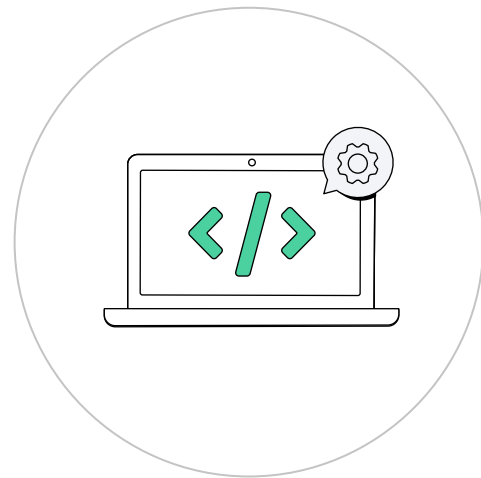
Метаданные ответа (Content-Type, Server, Date и др.)

## 4 Пустая строка

Разделяет заголовки и тело

## 5 Тело ответа (Body)

Данные (HTML, JSON, XML и др.), если есть



# Что означают эти ответы?

Напишите ваши варианты в чат



1

```
HTTP/1.1 204 No Content
Date: Mon, 01 Jul 2024 12:00:00 GMT
Server: Apache
```

2

```
HTTP/1.1 401 Unauthorized
Date: Mon, 01 Jul 2024 12:00:00 GMT
Server: Apache
WWW-Authenticate: Bearer realm="API"
Content-Type: application/json

{
  "error": "Unauthorized",
  "message": "Missing or invalid API key"
}
```

3

```
HTTP/1.1 500 Internal Server
Date: Mon, 01 Jul 2024 12:00:00 GMT
Server: Apache
Content-Type: application/json

{
  "error": "Internal Server Error",
  "message": "Database connection failed"
}
```

# Пример HTTP ответа

Стандартный успешный ответ (200 OK)

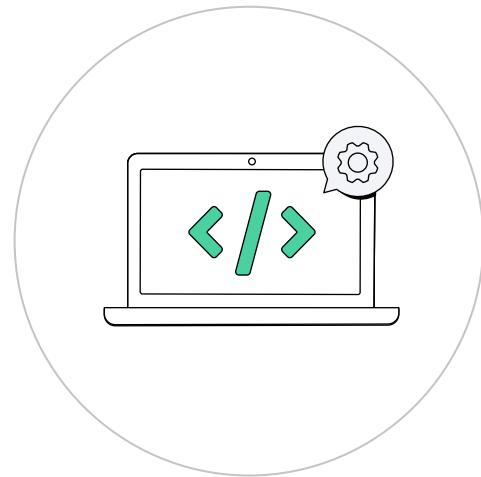
```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Mon, 15 Apr 2024 12:00:00 GMT
Server: Apache/2.4.41
Content-Length: 42
```

```
{
  "status": "success",
  "data": {
    "id": 123,
    "name": "Example Product"
  }
}
```



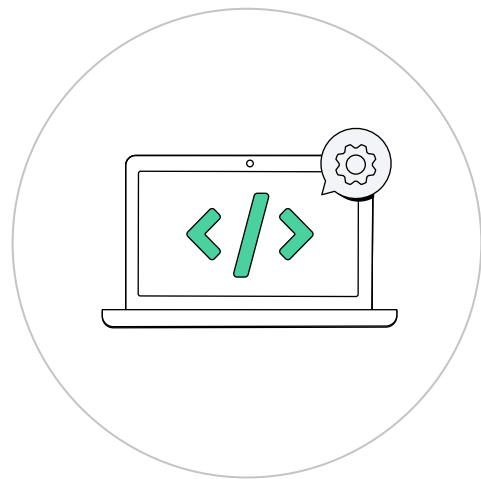
# Коды статусов HTTP

- **1xx** – информационные ответы (102, 103).
- **2xx** – успешные ответы (200, 201, 204).
- **3xx** – перенаправления (301, 302, 304).
- **4xx** – ошибки клиента (400, 403, 404).
- **5xx** – ошибки сервера (500, 502, 503).



# Популярные коды

- **200 OK** — всё хорошо
- **201 Created** — ресурс успешно создан
- **204 No Content** — без тела (например, после удаления)
- **400 Bad Request** — ошибка в запросе
- **401 Unauthorized** — нужна авторизация
- **403 Forbidden** — доступ запрещён
- **404 Not Found** — ресурс не найден
- **500 Internal Server Error** — ошибка на сервере



# Пример HTTP ответа

Ответ с перенаправлением (301 Moved Permanently)

```
HTTP/1.1 301 Moved Permanently
```

```
Location: https://example.com/new-page
```

```
Content-Type: text/html
```

```
Date: Mon, 15 Apr 2024 12:00:00 GMT
```

```
Server: nginx
```

```
Content-Length: 178
```

```
<html>
```

```
  <head><title>301 Moved Permanently</title></head>
```

```
  <body>
```

```
    <h1>301 Moved Permanently</h1>
```

```
    <p>The resource has been moved to <a href="https://example.com/new-page">this URL</a>.</p>
```

```
  </body>
```

```
</html>
```

# Пример HTTP ответа

Ошибка клиента (404 Not Found)

```
HTTP/1.1 404 Not Found
Content-Type: text/html
Date: Mon, 15 Apr 2024 12:00:00 GMT
Server: Microsoft-IIS/10.0
Content-Length: 127

<html>
  <head><title>404 Not Found</title></head>
  <body>
    <h1>404 Not Found</h1>
    <p>The requested resource was not found.</p>
  </body>
</html>
```

# Пример HTTP ответа

Ошибка сервера (500 Internal Server Error)

```
HTTP/1.1 500 Internal Server Error
```

```
Content-Type: application/json
```

```
Date: Mon, 15 Apr 2024 12:00:00 GMT
```

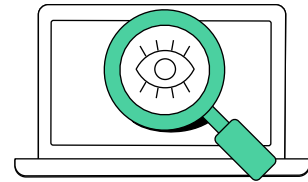
```
Server: cloudflare
```

```
Content-Length: 65
```

```
{  
  "error": "Internal Server Error",  
  "message": "Something went wrong on our side."  
}
```

# Демонстрация работы

Postman



# Библиотека requests



8

# Выполнение простого запроса

Выполнение простого GET-запроса:

```
import requests

url = "https://httpbin.org/get" resp = requests.get(url)
print(resp)
> '<Response [200]>'

print(resp.status_code)
> 200
```

Чтобы сделать POST-запрос, используйте функцию **post**:

```
requests.post(url)
```



# Атрибуты и методы объекта Response

- **status\_code** – HTTP-статус ответа
- **headers** – заголовки ответа
- **content** – содержимое ответа в байтах
- **text** – содержимое ответа в текстовом представлении (utf8, так как все строки в Python – юникодные)
- **json()** – представление ответа в виде словаря. Работает только в том случае, если сервер вернул валидный JSON. Используется при работе с API.

# Параметры запроса и заголовки

Чтобы передать параметры и заголовки в запросе, используйте именованные аргументы:

```
import requests

url = "https://httpbin.org/get"
params = {"foo": "bar", "message": "hello"}
headers = {"Authorization": "secret-token-123"}

resp = requests.get(url, params=params, headers=headers)
```

**Важно:** конкретные значения параметров и заголовков зависят от сервера, к которому происходит обращение.

Для того, чтобы узнать точные значения параметров, нужно читать документацию или проверять опытным путем.

# Передача тела запроса

Чтобы передать тело запроса, используется параметр **data** и заголовок **Content-Type** для уточнения формата данных:

```
import requests

url = "https://httpbin.org/post"
headers = {"Content-Type": "Application/json"}
payload = {"a": 1, "b": 2}

resp = requests.post(url, headers=headers, data=json.dumps(payload))
```

**Важно:** конкретный вид тела зависит от сервера, к которому происходит обращение. Для того чтобы узнать точные значения параметров, нужно читать документацию или проверять опытным путем.

# Передача тела запроса в формате JSON

Так как данные в REST передаются чаще всего в формате JSON, то и разработчики requests упростили нам эту задачу.

```
import requests

url = "https://httpbin.org/post"
payload = {"a": 1, "b": 2}

resp = requests.post(url, json=payload)
```

Если ты пишешь **json=data** — библиотека сама:

- сериализует словарь в JSON (`json.dumps(...)`)
- добавит заголовок `Content-Type: application/json`

**Важно:** конкретный вид тела зависит от сервера, к которому происходит обращение. Для того чтобы узнать точные значения параметров, нужно читать документацию или проверять опытным путем.

# Загрузка файлов

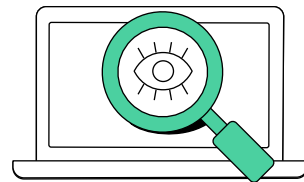
Чтобы передавать файл, его нужно открыть в байтовом режиме и передать объект файла в параметр files:

```
import requests

with open('gifs/your_file.mp4', 'rb') as f:
    resp = requests.post('http://httpbin.org/post', files={"file": f})
```

# Демонстрация работы

Работа с API Nasa и Яндекс-Диска



# Итоги и ваши вопросы



# Сегодня мы:



Познакомились с протоколом HTTP

- Клиент и сервер
- Что такое API
- Принципы HTTP



Рассмотрели из чего состоят запросы и ответы

- Методы
- URL
- Заголовки и параметры
- Тело



Узнали как работать с этим протоколом с помощью библиотеки requests

- Работа с документацией API
- Как отправлять запросы (параметры, заголовки, тело, файлы)
- Что можно узнать из ответа







**Ваши вопросы?**

# Работа с библиотекой requests, http запросы

Тимур Анвардинов  
Инженер по контролю качества, [smotreshka.tv](https://smotreshka.tv)

