# Понятное руководство по Requests и JSON: Учебная шпаргалка для начинающего Python-программиста

### Введение: Наш путь в мир веб-данных

Добро пожаловать в увлекательный мир сетевого программирования на Python! Если вы уже знакомы с PySide, то вы знаете, что такое взаимодействие с пользователем. Теперь пришло время научиться взаимодействовать с веб-сервисами и получать данные из интернета. Может показаться, что понятия JSON и requests сложны и запутаны, но на самом деле они очень логичны.

Представим, что ваш Python-скрипт — это посетитель в ресторане, который хочет заказать еду.

* **Вы** (ваш скрипт) — клиент.
* **Интернет** — это официант.
* **Веб-сервер** — это кухня, где готовят еду.
* **HTTP-запрос** — это ваш заказ, который вы передаете официанту.
* **HTTP-ответ** — это готовое блюдо, которое официант приносит вам обратно.
* **JSON** — это способ записи заказа и меню, понятный и вам, и повару. Он гарантирует, что вы получите именно то, что заказывали.

Этот отчет — ваша подробная инструкция, которая поможет разобраться в каждом из этих шагов, от основ до профессиональной обработки возможных ошибок.

### Часть 1: Фундамент — Что такое JSON и HTTP?

Прежде чем начать писать код, необходимо заложить прочную основу, понимая, как устроена коммуникация в интернете и в каком формате передаются данные.

#### 1.1. Основы веб-коммуникации: Цикл HTTP-запрос/ответ

HTTP (Hypertext Transfer Protocol) — это базовый протокол, или «язык», на котором общаются клиент и сервер в интернете.1 Процесс взаимодействия всегда состоит из двух частей: клиент отправляет

**HTTP-запрос** к серверу, а сервер формирует и отправляет **HTTP-ответ**.2

Каждый запрос содержит несколько ключевых компонентов:

* **Метод (Verb):** Это своего рода "команда" для сервера, которая определяет, что вы хотите сделать. Самые распространенные методы — это GET и POST. Метод GET используется, когда нужно просто получить какие-либо данные с сервера, например, загрузить страницу или список новостей.2 Метод  
  POST применяется, когда вы отправляете новые данные на сервер, например, заполняете форму регистрации или создаете новый ресурс.2
* **URL:** Это уникальный адрес ресурса, к которому вы обращаетесь.2
* **Заголовки (Headers):** Это дополнительная информация о запросе. Заголовки могут содержать метаданные, например, информацию об авторизации.2

После получения запроса, сервер обрабатывает его и отправляет клиенту ответ. Ответ также состоит из нескольких важных частей:

* **Статусный код (Status Code):** Это самое важное, что говорит нам сервер о результате обработки запроса. Статусный код — это трехзначное число, которое однозначно указывает, успешен ли запрос или произошла какая-то ошибка.2 Коды из диапазона 200-299 означают успех.6
* **Заголовки (Headers):** Эти заголовки содержат метаданные об ответе, например, тип данных в теле ответа.2
* **Тело (Body):** Это основное содержимое ответа. Если вы запрашивали веб-страницу, то в теле будет HTML-код; если вы обменивались данными с API, то, скорее всего, в теле будет JSON-строка.2

Для начинающего программиста критически важно понимать значение основных статусных кодов, поскольку они мгновенно сообщают, что пошло не так. Например, код 404 означает, что запрошенный ресурс не найден, а 200 — что всё прошло успешно.5

**Таблица 1: Важнейшие коды состояния HTTP**

|  |  |  |
| --- | --- | --- |
| Код состояния | Название | Простое объяснение |
| 200 | OK | Запрос успешно выполнен. Всё отлично, данные получены. |
| 201 | Created | Запрос успешно выполнен, и на сервере создан новый ресурс (например, после POST-запроса).7 |
| 400 | Bad Request | Запрос сформирован неправильно или данные не прошли проверку.7 |
| 401 | Unauthorized | Требуется аутентификация для доступа к ресурсу.7 |
| 404 | Not Found | Запрашиваемый ресурс (URL) не существует на сервере.5 |
| 500 | Internal Server Error | На сервере произошла внутренняя ошибка.7 |

#### 1.2. JSON: формат данных для всего мира

JSON (JavaScript Object Notation) — это легковесный формат для обмена данными, основанный на тексте.8 Он невероятно популярен, потому что он прост, понятен для чтения человеком и, что самое главное, его легко обрабатывать программами.10

Для пользователя, который уже знаком со словарями и списками в Python, есть критически важное сопоставление: **JSON — это просто ваш знакомый словарь, но в виде обычной текстовой строки**.10 Это знание мгновенно снимает барьер и позволяет без труда перейти от концепции к коду. JSON-объект с парами "ключ-значение" напрямую сопоставляется с

dict в Python, а JSON-массив — со списком (list).10

**Таблица 2: Сравнение типов данных JSON и Python**

|  |  |  |
| --- | --- | --- |
| Тип данных JSON | Аналог в Python | Пример |
| object | dict (словарь) | {'key': 'value'} |
| array | list (список) | ['apple', 'banana', 'cherry'] |
| string | str (строка) | 'hello' |
| number | int или float | 123 или 3.14 |
| true/false | True/False | True |
| null | None | None |

Для работы с этим форматом в Python используется встроенный модуль json.11 У него есть две основные функции:

* **Сериализация (json.dumps()):** Этот процесс превращает объект Python (например, словарь) в JSON-строку. Это необходимо, чтобы подготовить данные для отправки по сети, так как по интернету можно передавать только текст или байты.10  
  Python  
  import json  
    
  python\_dict = {'name': 'Alice', 'age': 30, 'is\_student': False}  
  # Превращаем словарь в JSON-строку  
  json\_str = json.dumps(python\_dict, indent=4) # indent=4 для красивого форматирования  
    
  print(json\_str)  
  # Вывод:  
  # {  
  # "name": "Alice",  
  # "age": 30,  
  # "is\_student": false  
  # }
* **Десериализация (json.loads()):** Обратный процесс, который превращает JSON-строку, полученную по сети, обратно в объект Python (словарь или список), с которым удобно работать в вашем коде.10

### Часть 2: Практика — Работаем с библиотекой Requests

Библиотека requests — это стандарт де-факто для выполнения HTTP-запросов в Python. Она намного удобнее и проще, чем встроенная urllib, и даже официальная документация Python рекомендует использовать именно её.3

#### 2.1. Ваш первый шаг: Установка и GET-запрос

Поскольку requests не входит в стандартную библиотеку Python, её нужно установить с помощью пакетного менеджера pip.13

Bash

pip install requests

После установки можно приступать к работе. Чтобы выполнить GET-запрос и получить данные, нужно всего лишь вызвать метод requests.get() с URL-адресом.4

Python

import requests  
  
# Выполняем GET-запрос к публичному API GitHub  
response = requests.get('https://api.github.com/events')  
print(f"Статусный код: {response.status\_code}") # Проверяем, что ответ успешен (200)

Зачастую в GET-запросе нужно передавать дополнительные параметры, например, для поиска. В этом случае requests избавляет вас от необходимости вручную форматировать URL.4 Вы просто передаете словарь с параметрами в аргументе

params 4:

Python

import requests  
  
# Параметры для запроса  
payload = {'key1': 'value1', 'key2': 'value2'}  
  
# Requests автоматически преобразует это в URL: httpbin.org/get?key1=value1&key2=value2  
response = requests.get('https://httpbin.org/get', params=payload)  
print(response.url)

#### 2.2. Отправка данных: POST-запрос с JSON

Для отправки данных используется метод requests.post().15 Здесь есть важный нюанс, который часто сбивает с толку новичков.

Вы можете столкнуться с двумя основными способами передачи данных:

1. **Форм-данные (data):** Использование параметра data для отправки данных в виде HTML-формы. Библиотека автоматически кодирует ваш словарь в формат application/x-www-form-urlencoded.15
2. **JSON-данные (json):** Отправка данных в формате JSON. Это самый распространенный способ для взаимодействия с современными API.15

Для новичка это может показаться сложным, но requests предоставляет элегантное решение. Вместо того, чтобы вручную сериализовать словарь в JSON-строку с помощью json.dumps() и затем устанавливать заголовок Content-Type, можно использовать специальный аргумент json.13

Использование requests.post(url, json=data) — это не просто синтаксический сахар. Это решение одной из самых распространенных проблем у начинающих. Библиотека **автоматически** выполняет две критически важные операции:

1. Сериализует ваш Python-словарь в JSON-строку.
2. Устанавливает заголовок Content-Type: application/json, который сообщает серверу, что данные приходят в формате JSON.15

Именно эта автоматизация помогает избежать ошибок и значительно упрощает процесс отправки данных.

Python

import requests  
  
url = 'https://jsonplaceholder.typicode.com/posts'  
# Данные, которые мы хотим отправить  
new\_data = {  
 "userID": 1,  
 "title": "Making a POST request",  
 "body": "This is the data we created."  
}  
  
# Отправляем POST-запрос, используя аргумент json  
response = requests.post(url, json=new\_data)  
# Ответ сервера будет содержать созданный ресурс  
print(response.json())

### Часть 3: Понимаем ответ — Объект Response

После выполнения любого запроса (get, post и т.д.) библиотека requests возвращает специальный объект Response.4 Этот объект содержит всю информацию, которую сервер прислал в ответ.14

#### 3.1. Содержимое объекта Response

Самые полезные атрибуты объекта Response:

* response.status\_code: Числовой код состояния HTTP (например, 200, 404).16
* response.text: Содержимое ответа в виде строки. Библиотека делает "обоснованные предположения" о кодировке на основе HTTP-заголовков.4
* response.content: Содержимое ответа в виде байтов. Это полезно для работы с бинарными данными, такими как изображения или архивы.4
* response.headers: Словарь с заголовками ответа.

#### 3.2. Превращаем JSON-ответ в словарь Python с response.json()

Большинство современных API возвращают данные в формате JSON. Самый удобный способ получить эти данные в виде словаря или списка Python — это использовать метод response.json().13

Этот метод делает то, что мы обсуждали ранее: он десериализует (парсит) JSON-строку из тела ответа и превращает её в объект Python, с которым вы можете работать как с обычным словарем.14

Python

import requests  
  
url = "https://jsonplaceholder.typicode.com/posts/1"  
response = requests.get(url)  
  
# Превращаем JSON-ответ в словарь Python  
data = response.json()  
print(data['title']) # Доступ к ключам, как в обычном словаре

Однако этот метод может вызвать ошибку JSONDecodeError, если ответ сервера не содержит валидного JSON, например, если это пустая строка или HTML-код ошибки. Это происходит, когда запрос завершился неудачей, и сервер вернул не те данные, которые вы ожидали.13 Именно поэтому перед тем, как вызывать

.json(), необходимо всегда проверять, что запрос был успешным.

### Часть 4: Профессиональный подход — Обработка ошибок

В мире веб-программирования всегда нужно быть готовым к тому, что что-то может пойти не так. Ваш запрос может не дойти из-за отсутствия интернета, сервер может быть недоступен или вернуть ошибку.16 Надежный код всегда умеет gracefully обрабатывать такие ситуации.

#### 4.1. Почему код может сломаться и что с этим делать?

Распространенные ошибки:

* **Ошибки соединения (ConnectionError):** Нет подключения к интернету, или домен не существует.16
* **Таймаут (Timeout):** Сервер не ответил в течение установленного времени.16
* **Ошибки HTTP (HTTPError):** Сервер ответил, но вернул статусный код из диапазона 4xx или 5xx (например, 404, 500).6

#### 4.2. Простое решение: response.raise\_for\_status()

Чтобы ваш код не продолжал работать, как ни в чём не бывало, когда запрос завершился ошибкой, requests предоставляет удобный метод response.raise\_for\_status().6

Этот метод проверяет статусный код:

* Если код в диапазоне 200-299, он ничего не делает.
* Если код 4xx или 5xx, он **автоматически вызывает исключение requests.exceptions.HTTPError**.6

Это предотвращает так называемые "тихие сбои" и является отличной практикой для новичка.

Python

import requests  
  
# Несуществующий URL, который вернет 404  
try:  
 response = requests.get('https://example.com/bad-url')  
 # Если здесь будет ошибка 404, то raise\_for\_status() вызовет исключение  
 response.raise\_for\_status()   
 print("Запрос успешен!") # Эта строка не выполнится  
except requests.exceptions.HTTPError as err:  
 print(f"Произошла HTTP-ошибка: {err}")  
 # Вывод: Произошла HTTP-ошибка: 404 Client Error: Not Found for url: https://example.com/bad-url

#### 4.3. Структура try-except для надежности

Наиболее надежный подход — использовать блоки try-except для обработки различных видов исключений, которые могут возникнуть при запросе.16

requests имеет иерархию исключений, где requests.exceptions.RequestException является родительским классом для всех остальных. Это позволяет вам либо отлавливать все ошибки сразу, либо работать с ними по отдельности.16

Python

import requests  
  
url = 'http://api.gle.com/data' # Пример URL, который не существует  
try:  
 response = requests.get(url, timeout=5) # Установка таймаута  
 response.raise\_for\_status() # Вызовет HTTPError для 4xx/5xx кодов  
 data = response.json()  
 print("Данные успешно получены!")  
   
except requests.exceptions.HTTPError as errh:  
 print(f"Ошибка HTTP: {errh}")  
except requests.exceptions.ConnectionError as errc:  
 print(f"Ошибка соединения: {errc}")  
except requests.exceptions.Timeout as errt:  
 print(f"Истекло время ожидания: {errt}")  
except requests.exceptions.RequestException as err:  
 print(f"Произошла какая-то другая ошибка: {err}")

**Таблица 3: Основные исключения в Requests**

|  |  |  |
| --- | --- | --- |
| Исключение | Что оно означает | Пример причины |
| HTTPError | Ошибка, связанная со статусным кодом (4xx, 5xx) | URL не существует (404), ошибка на сервере (500).6 |
| ConnectionError | Ошибка, связанная с сетевым подключением | Нет доступа в интернет, неверный домен.16 |
| Timeout | Истекло время ожидания ответа от сервера | Сервер очень медленно отвечает.16 |
| RequestException | Родительский класс для всех исключений requests | Используется для общего отлова любой ошибки, связанной с запросом.16 |

### Заключение и учебная шпаргалка

HTTP-протокол — это механизм "запрос-ответ", а JSON — это просто способ представить данные в виде текста, очень похожего на словарь Python. Библиотека requests делает работу с этим протоколом невероятно простой и интуитивно понятной. Использование response.json() для парсинга данных и блоков try-except с response.raise\_for\_status() для обработки ошибок — это залог написания надежного и профессионального кода.

Сохраните или распечатайте эту шпаргалку, чтобы она всегда была под рукой.

#### Учебная шпаргалка для распечатки

1. Установка библиотеки Requests

python -m pip install requests

**2. Фундамент: HTTP и JSON**

* **HTTP:** Клиент отправляет **запрос** к серверу, сервер отправляет **ответ**.
* **Методы:** GET (получить), POST (отправить).
* **Коды:** 200 (успех), 404 (не найден), 500 (ошибка сервера).
* **JSON:** Формат данных, похожий на словари и списки Python, но в виде текста.

**3. Основы работы с Requests**

* **Импорт:** import requests
* **GET-запрос:** response = requests.get('https://api.github.com/events')
* **GET-запрос с параметрами:**  
  Python  
  params = {'query': 'python'}  
  response = requests.get('https://example.com/search', params=params)
* **POST-запрос с JSON (рекомендуемый способ):**  
  Python  
  data = {'title': 'My new post'}  
  response = requests.post('https://jsonplaceholder.typicode.com/posts', json=data)

**4. Работа с объектом Response**

* **Проверка статуса:** response.status\_code (например, 200)
* **Получение данных как словаря:** data = response.json()
* **Получение текста:** html\_content = response.text

**5. Надежная обработка ошибок**

* **Автоматическая проверка статуса:** response.raise\_for\_status() — вызывает ошибку, если код 4xx или 5xx.
* **Базовый шаблон try-except:**  
  Python  
  import requests  
    
  try:  
   response = requests.get('https://example.com/api', timeout=5)  
   response.raise\_for\_status()  
   data = response.json()  
   print("Данные успешно получены!")  
  except requests.exceptions.RequestException as err:  
   print(f"Произошла ошибка: {err}")

#### Источники

1. How the Web Works, HTTP Request/Response Cycle, дата последнего обращения: сентября 22, 2025, <https://backend.turing.edu/module2/lessons/how_the_web_works_http>
2. Что такое HTTP-запрос и HTTP-ответ? - Узнай ответ в ... - Яндекс, дата последнего обращения: сентября 22, 2025, <https://yandex.kz/yanswers/c/tehnologiyalar/q/chto-takoe-http-zaproc-i-http-otvet-2596189833>
3. Модуль Requests Python: что это такое, как установить ... - Skypro, дата последнего обращения: сентября 22, 2025, <https://sky.pro/media/modul-requests-v-python/>
4. Quickstart — Requests 2.32.5 documentation, дата последнего обращения: сентября 22, 2025, <https://requests.readthedocs.io/en/master/user/quickstart/>
5. medium.com, дата последнего обращения: сентября 22, 2025, <https://medium.com/@jen_strong/the-request-response-cycle-of-the-web-1b7e206e9047#:~:text=An%20HTTP%20response%20is%20what,request%20would%20be%20%E2%80%9C404%E2%80%9D.>
6. response.raise\_for\_status() - Python requests - GeeksforGeeks, дата последнего обращения: сентября 22, 2025, <https://www.geeksforgeeks.org/python/response-raise_for_status-python-requests/>
7. Гайд по протоколу HTTP: Структура, Расшифровка и Механизм ..., дата последнего обращения: сентября 22, 2025, <https://cloud.ru/blog/protokol-http>
8. www.mongodb.com, дата последнего обращения: сентября 22, 2025, <https://www.mongodb.com/resources/languages/what-is-json#:~:text=JSON%20is%20a%20text%2Dbased%2C%20light%2Dweight%20data%20interchange,%E2%80%9D%20%3A%20%E2%80%9C%2B52%E2%80%9D%20%7D>
9. What Is JSON? | A Beginner's Guide - MongoDB, дата последнего обращения: сентября 22, 2025, <https://www.mongodb.com/resources/languages/what-is-json>
10. JSON в Python: как работать, как открыть, прочитать, записать ..., дата последнего обращения: сентября 22, 2025, <https://sky.pro/media/modul-json-v-python/>
11. Что такое json в Python? Онлайн справочник на itProger, дата последнего обращения: сентября 22, 2025, <https://itproger.com/spravka/python/json>
12. Библиотека Requests для Python: код и практика - Skillbox, дата последнего обращения: сентября 22, 2025, <https://skillbox.ru/media/code/biblioteka-requests-dlya-python-kod-i-praktika/>
13. Getting Started with Python HTTP Requests for REST APIs - DataCamp, дата последнего обращения: сентября 22, 2025, <https://www.datacamp.com/tutorial/making-http-requests-in-python>
14. response.json() - Python requests - GeeksforGeeks, дата последнего обращения: сентября 22, 2025, <https://www.geeksforgeeks.org/python/response-json-python-requests/>
15. Python Requests - How to Send POST Requests | ScrapeOps, дата последнего обращения: сентября 22, 2025, <https://scrapeops.io/python-web-scraping-playbook/python-requests-post-requests/>
16. Exception Handling Of Python Requests Module - GeeksforGeeks, дата последнего обращения: сентября 22, 2025, <https://www.geeksforgeeks.org/python/exception-handling-of-python-requests-module/>
17. Response: json() method - Web APIs | MDN - Mozilla, дата последнего обращения: сентября 22, 2025, <https://developer.mozilla.org/en-US/docs/Web/API/Response/json>
18. Intercept exception handling - Python Discussions, дата последнего обращения: сентября 22, 2025, <https://discuss.python.org/t/intercept-exception-handling/25823>