

# ЗАГРУЗКА ДАННЫХ. API ВЕБ-ПРИЛОЖЕНИЙ

---

ФАЙЛЫ ФОРМАТА JSON

ВЫЗОВЫ API

ВИЗУАЛИЗАЦИЯ ДАННЫХ ВЫЗОВА API

Гаврилов Денис Андреевич, преподаватель кафедры СИ ФИТ НГУ

Формат JSON.

JSON в Python.

Взаимное преобразование  
типов данных Python и JSON.

Чтение и запись файлов JSON.

# ФАЙЛЫ ФОРМАТА JSON

# ФОРМАТ JSON

---

**JSON (JavaScript Object Notation)** - это текстовый формат файла, применяющийся, в основном, для передачи данных между сервером и веб-приложением.

JSON построен на двух структурах:

- Набор пар «имя-значение», которые могут быть реализованы как объект, запись, словарь, хеш-таблица, список «ключей-значений» или ассоциативный массив;
- Упорядоченный список значений, реализованный в виде массива, вектора, списка или последовательности.

```

{
  "Comment":"My comment",
  "Count":10,

  "DiskParam":
  {
    "DB":10.000000,
    "DBAngle":1.234000
  },

  "Blades":
  [
    {
      "Caption":"A",
      "Value":65
    },
    {
      "Caption":"B",
      "Value":66
    },
    {
      "Caption":"C",
      "Value":67
    }
  ],

  "Slots":
  [
    0,1,2
  ]
}

```

# Единичные атрибуты

# Словарь

# Список словарей

# Список значений

# ФОРМАТ JSON: ПРИМЕР

# JSON В PYTHON

---

В Python для поддержки формата JSON используется модуль *json*.

Преобразование (чтение) данных выполняется при помощи метода *json.loads()*:

```
import json
```

```
to_python = json.loads(my_json_string)
```

```
print(to_python['Blades'])
```

```
print(to_python)
```

# Преобразование структур JSON в типы данных Python

```
# [{'Caption': 'A', 'Value': 65}, {'Caption': 'B', 'Value': 66},  
#   {'Caption': 'C', 'Value': 67}];
```

# JSON B PYTHON

---

В Python для поддержки формата JSON используется модуль *json*.

Преобразование данных в JSON (запись) выполняется при помощи метода *json.dumps()*:

```
import json

blade = [{'Caption': 'A', 'Value': 65},
         {'Caption': 'B', 'Value': 66},
         {'Caption': 'C', 'Value': 67}]

file = {'Comment': 'My comment', 'Blade': blade}

to_json = json.dumps(file)

print(to_json)
```

```
# {"Comment": "My comment",
# "Blade": [{"Caption": "A", "Value": 65},
# {"Caption": "B", "Value": 66}, {"Caption": "C", "Value": 67}] }
```

# ФУНКЦИИ ЧТЕНИЯ И ЗАПИСИ JSON

---

**Важно:** функция `json.load()` используется для чтения (преобразования) файла, а `json.loads()` – для чтения (преобразования) строки.

(*loads* расшифровывается как «*load string*»)

Аналогично, `json.dump()` применяется, если нужно записать данные в файл JSON, а `json.dumps()` – если данные нужно представить в виде JSON-строки.

# ЗАПИСЬ ФАЙЛА JSON

---

Пример записи файла JSON с использованием *json.dump()*.

```
blade = [{'Caption': 'A', 'Value': 65},
         {'Caption': 'B', 'Value': 66},
         {'Caption': 'C', 'Value': 67}]

file = {'Comment': 'My comment', 'Blade': blade}

with open('test_file.json', 'w') as f:
    json.dump(file, f)
```

# test\_file.json:

```
{
  "Comment": "My comment",
  "Blade":
  [
    {
      "Caption": "A", "Value": 65
    },
    {
      "Caption": "B", "Value": 66
    },
    {
      "Caption": "C", "Value": 67
    }
  ]
}
```



# ЧТЕНИЕ ФАЙЛА JSON

---

Пример чтения файла JSON с использованием *json.load()*.

```
blade = [{'Caption': 'A', 'Value': 65},
         {'Caption': 'B', 'Value': 66},
         {'Caption': 'C', 'Value': 67}]

file = {'Comment': 'My comment', 'Blade': blade}

with open('test_file.json', 'w') as f:
    json.dump(file, f)

with open('test_file.json', 'r') as f:
    json_data = json.load(f)

print(json_data['Blade'])
```

```
# [{"Caption": "A", "Value": 65}, {"Caption": "B", "Value": 66},
#  {"Caption": "C", "Value": 67}] }
```

# ДВОЙНАЯ ЗАПИСЬ

**Важно:** избегайте «двойной записи» файла JSON.

После вызова `json.dumps()` мы получаем строку, форматированную в соответствии с правилами JSON;

Последующий вызов `json.dump()` предпримет повторную попытку форматирования, воспринимая уже готовую строку как элемент данных.

# Ошибка

# test\_file.json: "{\\"Blade\\": [{\\"Caption\\": \\"A\\",\\"Value\\": 65},... "

```
blade = [{'Caption': 'A', 'Value': 65},
          {'Caption': 'B', 'Value': 66},
          {'Caption': 'C', 'Value': 67}]
file = {'Comment': 'My comment', 'Blade': blade}
```

```
to_json = json.dumps(file)

with open('test_file.json', 'w') as f:
    json.dump(to_json, f)
```

```
with open('test_file.json', 'r') as f:
    json_data = json.load(f)

print(json_data['Blade'])
```

# ДВОЙНАЯ ЗАПИСЬ

**Важно:** избегайте «двойной записи» файла JSON.

После вызова `json.dumps()` мы получаем строку, сформатированную в соответствии с правилами JSON;

Последующий вызов `json.dump()` предпримет повторную попытку форматирования, воспринимая уже готовую строку как элемент данных.

# Корректная запись файла

```
blade = [{'Caption': 'A', 'Value': 65},
          {'Caption': 'B', 'Value': 66},
          {'Caption': 'C', 'Value': 67}]
file = {'Comment': 'My comment', 'Blade': blade}
```

```
to_json = json.dumps(file)

with open('test_file.json', 'w') as f:
    print(to_json, file=f)
```

```
with open('test_file.json', 'r') as f:
    json_data = json.load(f)

print(json_data['Blade'])
```

Вызовы API.  
Библиотека requests.

Построение простого запроса.

Конечные точки вызова.

Работа с ответом.

# ВЫЗОВЫ API

# ВЫЗОВЫ API

---

**API (Application programming interface)** – способ (набор классов, процедур, функций, структур, констант...), посредством которого одна программа может взаимодействовать с другой.

Так, программа (например, на языке Python) может использовать API веб-приложения для запроса конкретной информации с сайта.

Далее, на основании загруженных данных можно будет построить визуализацию этих данных.

# ВЫЗОВЫ API

---

**API веб-приложения** – это часть веб-сайта, которая может взаимодействовать с программой посредством особым образом построенного url-адреса – для запроса информации.

Такой запрос называется вызовом API. Например:

```
response = requests.get('https://example.com')
```

Запрашиваемые данные возвращаются в удобном формате (например, CSV или JSON).

# БИБЛИОТЕКА REQUESTS

---

Для выполнения API-запросов в Python используется библиотека *requests*.

Данная библиотека предоставляет интерфейс для *синхронного* выполнения HTTP-запросов.

При работе в PyCharm, библиотека *requests* недоступна по умолчанию; её следует установить.

# ПОСТРОЕНИЕ ПРОСТОГО ЗАПРОСА

---

Построим простой запрос к сервису Github:

```
import requests

url = 'https://api.github.com'

r = requests.get(url)

print(f"Status code:{r.status_code}")

response_dict = r.json()

print(response_dict)
```

*https://github.com*

➤ Обратимся к базовому url-адресу API, находящемуся по адресу *https://api.github.com*.

**Важно:** не всякий адрес API будет начинаться с префикса *https://api...*



# ПОСТРОЕНИЕ ПРОСТОГО ЗАПРОСА

---

Обратимся к базовому url-адресу API, находящемуся по адресу *https://api.github.com*.

```
import requests

url = 'https://api.github.com'

r = requests.get(url)

print(f"Status code:{r.status_code}")

response_dict = r.json()

print(response_dict)
```

➤ Подключение библиотеки *requests*.

➤ Указание строки с адресом, по которому будет направлен запрос.

# ПОСТРОЕНИЕ ПРОСТОГО ЗАПРОСА

---

Обратимся к базовому url-адресу API, находящемуся по адресу *https://api.github.com*.

```
import requests

url = 'https://api.github.com'

r = requests.get(url)
print(f"Status code:{r.status_code}")

response_dict = r.json()

print(response_dict)
```

- Направление запроса по заданному адресу. Результат (ответ) возвращается функцией в форме объекта класса *Respond*.
- Печать *кода состояния HTTP* – атрибута класса *Respond*.  
(200 – в случае успеха)

# ПОСТРОЕНИЕ ПРОСТОГО ЗАПРОСА

---

Обратимся к базовому url-адресу API, находящемуся по адресу *https://api.github.com*.

```
import requests  
  
url = 'https://api.github.com'  
  
r = requests.get(url)  
  
print(f"Status code:{r.status_code}")
```

```
response_dict = r.json()  
  
print(response_dict)
```

➤ Преобразование ответа в словарь, согласно формату JSON.

➤ Вывод полученных данных на печать.

# КОНЕЧНЫЕ ТОЧКИ

---

```
import requests

url = 'https://api.github.com'

r = requests.get(url)

print(f"Status code:{r.status_code}")

response_dict = r.json()

print(response_dict)
```

Похожий результат можно получить, обратившись по адресу

*<https://api.github.com>*

напрямую из окна веб-браузера.

Данные адреса – адреса конечных точек («директорий») вызовов API.

```
# {'current_user_url': 'https://api.github.com/user',
# 'current_user_authorizations_html_url': 'https://github.com/settings/connections/applications{/client_id}', ..
```

# КОНЕЧНЫЕ ТОЧКИ

```
import requests

url = 'https://api.github.com'

r = requests.get(url)

print(f"Status code:{r.status_code}")

response_dict = r.json()

print(response_dict)
```

Похожий результат можно получить, обратившись по адресу

*<https://api.github.com>*

напрямую из окна веб-браузера.



The screenshot shows a web browser window with the address bar displaying `api.github.com`. The main content area shows a JSON object with various API endpoints for the current user, such as `current_user_url`, `current_user_authorizations_html_url`, `authorizations_url`, `code_search_url`, `commit_search_url`, `emails_url`, `emojis_url`, `events_url`, `feeds_url`, `followers_url`, `following_url`, `gists_url`, `hub_url`, `issue_search_url`, `issues_url`, `keys_url`, `label_search_url`, and `notifications_url`.

# КОНЕЧНЫЕ ТОЧКИ

The screenshot shows the GitHub Docs website. The browser address bar displays `docs.github.com/en/rest/overview/resources-in-the-rest-api`. The page header includes the GitHub Docs logo, navigation links for `REST API`, `Overview`, and `Resources in the REST API`, and dropdown menus for `Free, Pro, & Team` and `English`. A left sidebar contains a table of contents with sections: `OVERVIEW` (expanded), `REFERENCE`, and `GUIDES`. Under `OVERVIEW`, the items are: `Resources in the REST API` (active), `Media types`, `Other authentication methods`, `Troubleshooting`, `API previews`, `Libraries`, `OpenAPI description`, and `GitHub App-enabled endpoints`. The main content area features the title `Resources in the REST API` and a subtitle `Learn how to navigate the resources provided by the GitHub API.` Below this is a paragraph: `This describes the resources that make up the official GitHub REST API. If you have any problems or requests, please contact GitHub Support.` A section titled `Current version` follows, with text: `By default, all requests to https://api.github.com receive the v3 version of the REST API. We encourage you to explicitly request this version via the Accept header.` A code block shows `Accept: application/vnd.github.v3+json`. The bottom paragraph states: `For information about GitHub's GraphQL API, see the v4 documentation. For information about migrating to GraphQL, see "Migrating from REST."` On the right, an `In this article` section lists links: `Current version`, `Schema`, `Authentication`, `Parameters`, `Root endpoint`, `GraphQL global node IDs`, `Client errors`, `HTTP redirects`, `HTTP verbs`, `Hypermedia`, `Pagination`, `Rate limiting`, `User agent required`, `Conditional requests`, `Cross origin resource sharing`, `JSON-P callbacks`, and `Timezones`.

< > ↺ 🔒 docs.github.com/en/rest/overview/resources-in-the-rest-api

GitHub Docs

REST API / Overview / Resources in the REST API

Free, Pro, & Team English

← All products

REST API

OVERVIEW

- Resources in the REST API
- Media types
- Other authentication methods
- Troubleshooting
- API previews
- Libraries
- OpenAPI description
- GitHub App-enabled endpoints

REFERENCE

GUIDES

## Resources in the REST API

Learn how to navigate the resources provided by the GitHub API.

This describes the resources that make up the official GitHub REST API. If you have any problems or requests, please contact [GitHub Support](#).

### Current version

By default, all requests to `https://api.github.com` receive the `v3` version of the REST API. We encourage you to explicitly request this version via the `Accept` header.

```
Accept: application/vnd.github.v3+json
```

For information about GitHub's GraphQL API, see the `v4` documentation. For information about migrating to GraphQL, see "`Migrating from REST`."

**In this article**

- [Current version](#)
- [Schema](#)
- [Authentication](#)
- [Parameters](#)
- [Root endpoint](#)
- [GraphQL global node IDs](#)
- [Client errors](#)
- [HTTP redirects](#)
- [HTTP verbs](#)
- [Hypermedia](#)
- [Pagination](#)
- [Rate limiting](#)
- [User agent required](#)
- [Conditional requests](#)
- [Cross origin resource sharing](#)
- [JSON-P callbacks](#)
- [Timezones](#)

# ПОСТРОЕНИЕ ЗАПРОСА. АДРЕС

---

Обратимся по адресу, соответствующему конечной точке  
"repository\_search\_url".

```
import requests

url='https://api.github.com/search/repositories?q=language:python&sort=stars'

headers={'Accept':'application/vnd.github.v3+json'}

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

print(f"Status code:{r.status_code}")
```

```
# url='https://api.github.com/search/repositories?q=language:python&sort=stars'
```

```
"repository_search_url": "https://api.github.com/search/repositories?q={query}&page,per_page,sort,order}" 23
```

# ПОСТРОЕНИЕ ЗАПРОСА. АДРЕС

Обратимся по адресу, соответствующему конечной точке  
"repository\_search\_url".

```
import requests
```

```
url='https://api.github.com/search/repositories?q=language:python&sort=stars'
```

```
headers={'Accept':'application/vnd.github.v3+json'}
```

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

```
print(f"Status code:{r.status_code}")
```

➤ Данная точка соответствует результату поиска по репозиториям сервиса Github.

➤ *q=language:python&sort=stars* – параметры поиска;

➤ Нам интересны данные репозитория, содержащие код на Python; результаты поиска – отсортированные по популярности.

```
# url='https://api.github.com/search/repositories?q=language:python&sort=stars'
```

```
"repository_search_url": "https://api.github.com/search/repositories?q={query}&page,per_page,sort,order}" 24
```



# ПОСТРОЕНИЕ ЗАПРОСА. ЗАГОЛОВКИ

Обратимся по адресу, соответствующему конечной точке

"repository\_search\_url".

```
import requests
```

```
url='https://api.github.com/search/repositories?q=language:python&sort=stars'
```

```
headers={'Accept':'application/vnd.github.v3+json'}
```

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

```
print(f"Status code:{r.status_code}")
```

➤ HTTP-заголовки (headers) используются для определения ряда параметров, управляющих запросами и ответами.

➤ Данный заголовок определяет версию REST API – v3;

➤ А также формат – JSON – ответа.

```
# url='https://api.github.com/search/repositories?q=language:python&sort=stars'
```

```
"repository_search_url": "https://api.github.com/search/repositories?q={query}&page,per_page,sort,order}" 25
```

# ПОСТРОЕНИЕ ЗАПРОСА. ЗАГОЛОВКИ

The screenshot shows the GitHub REST API documentation page. The browser address bar displays `docs.github.com/en/rest/overview/resources-in-the-rest-api`. The page header includes the GitHub Docs logo, navigation links for REST API, Overview, and Resources in the REST API, and options for Free, Pro, & Team plans and English language. The left sidebar contains a table of contents with sections: OVERVIEW (expanded), REST API, REFERENCE, and GUIDES. The main content area is titled 'Resources in the REST API' and includes a sub-header 'Current version' with a highlighted text box containing the `Accept: application/vnd.github.v3+json` header. A right sidebar lists links for 'In this article'.

docs.github.com/en/rest/overview/resources-in-the-rest-api

GitHub Docs

REST API / Overview / Resources in the REST API

Free, Pro, & Team English

## Resources in the REST API

Learn how to navigate the resources provided by the GitHub API.

This describes the resources that make up the official GitHub REST API. If you have any problems or requests, please contact [GitHub Support](#).

### Current version

By default, all requests to `https://api.github.com` receive the **v3** version of the REST API. We encourage you to explicitly request this version via the `Accept` header.

```
Accept: application/vnd.github.v3+json
```

For information about GitHub's GraphQL API, see the [v4 documentation](#). For information about migrating to GraphQL, see "[Migrating from REST](#)."

#### In this article

- [Current version](#)
- [Schema](#)
- [Authentication](#)
- [Parameters](#)
- [Root endpoint](#)
- [GraphQL global node IDs](#)
- [Client errors](#)
- [HTTP redirects](#)
- [HTTP verbs](#)
- [Hypermedia](#)
- [Pagination](#)
- [Rate limiting](#)
- [User agent required](#)
- [Conditional requests](#)
- [Cross origin resource sharing](#)
- [JSON-P callbacks](#)
- [Timezones](#)

# ПОСТРОЕНИЕ ЗАПРОСА. ЗАГОЛОВКИ

---

Несколько примеров заголовков:

HTTP Header	Описание
Accept	Какой тип контента может принять клиент
Content-Type	Какой тип контента в ответе сервера
User-Agent	Какое программное обеспечение клиент использует для связи с сервером
Server	Какое программное обеспечение сервер использует для связи с клиентом
Authentication	Кто вызывает API и с какими учетными данными

# ПОСТРОЕНИЕ ЗАПРОСА. РАБОТА С ОТВЕТОМ

---

Преобразование ответа к виду словаря; печать значений  
ключей получившегося словаря:

```
...  
r=requests.get(url, headers=headers)  
  
print(f"Status code:{r.status_code}")
```

```
response_dict=r.json()  
print(response_dict.keys())
```

```
repo_dicts=response_dict['items']  
print(f"Repositories returned:{len(repo_dicts)}")
```

```
# dict_keys(['total_count', 'incomplete_results', 'items'])
```

➤ Таким образом, имеем следующие структуры данных в  
ответе:

- Общее количество результатов поиска (ок. 8000000);
- Неполные результаты поиска;
- Полные результаты поиска

# ПОСТРОЕНИЕ ЗАПРОСА. РАБОТА С ОТВЕТОМ

---

Подсчет количества фактически возвращенных (в составе ответа) результатов поиска:

# Repositories returned:30

➤ 30 соответствует значению результатов, отображаемых на странице поиска по умолчанию.

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

```
print(f"Status code:{r.status_code}")
```

```
response_dict=r.json()  
print(response_dict.keys())
```

```
repo_dicts=response_dict['items']  
print(f"Repositories returned:{len(repo_dicts)}")
```

Формирование вызова.

Подготовка данных:

Работа с ответом вызова.

Визуализация данных.

# ВИЗУАЛИЗАЦИЯ ДАННЫХ, ПОЛУЧЕННЫХ В РЕЗУЛЬТАТЕ ВЫЗОВА API

# ВЫЗОВ API

---

```
from matplotlib import pyplot as plt
import requests

url='https://api.github.com/search/repositories?q=language:python&sort=stars&per_page=20'

headers={'Accept':'application/vnd.github.v3+json'}

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

print(f"Status code:{r.status_code}")
```

Обратимся по адресу, соответствующему конечной точке "repository\_search\_url".

```
# url='https://api.github.com/search/repositories?q=language:python&sort=stars&per_page=20'
```

```
"repository_search_url": "https://api.github.com/search/repositories?q={query}&page,per_page,sort,order}" 31
```

# ВИЗУАЛИЗАЦИЯ: ПОДГОТОВКА ДАННЫХ

---

```
...  
response_dict=r.json()  
repo_dicts=response_dict['items']  
print(f"Repositories returned:{len(repo_dicts)}")
```

➤ Сохранение результатов поиска для последующей обработки;

```
repo_names, stars=[],[]  
for repo_dict in repo_dicts:  
    repo_names.append(repo_dict['name'])  
    stars.append(repo_dict['stargazers_count'])
```

➤ Подсчет количества результатов поиска.

# Repositories returned:20



# ВИЗУАЛИЗАЦИЯ: ПОДГОТОВКА ДАННЫХ

---

Формирование данных для визуализации:

```
...
response_dict=r.json()

repo_dicts=response_dict['items']
print(f"Repositories returned:{len(repo_dicts)}")

repo_names, stars=[],[]
for repo_dict in repo_dicts:
    repo_names.append(repo_dict['name'])
    stars.append(repo_dict['stargazers_count'])
```

- repo\_names – содержит названия репозиториев, полученных в качестве ответа;
- stars – содержит числа звезд (меру «популярности») репозиториев, полученных в качестве ответа;

# ВИЗУАЛИЗАЦИЯ

---

Визуализация данных:

```
...  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
  
ax.set_title('Most-Starred Python Projects in GitHub')  
ax.set_xlabel('Project', fontsize=16)  
ax.set_ylabel('Stars', fontsize=16)  
fig.autofmt_xdate()  
  
ax.plot(repo_names, stars, c='red')  
  
plt.show()
```

- Установка встроенного стиля графика;
- Определение рисунка (figure) и набора графиков, содержащихся на данном рисунке (plots);

# ВИЗУАЛИЗАЦИЯ

---

Визуализация данных:

```
...  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
  
ax.set_title('Most-Starred Python Projects in GitHub')  
ax.set_xlabel('Project', fontsize=16)  
ax.set_ylabel('Stars', fontsize=16)  
fig.autofmt_xdate()  
  
ax.plot(repo_names, stars, c='red')  
  
plt.show()
```

➤ Установка параметров рисунка:

➤ название;

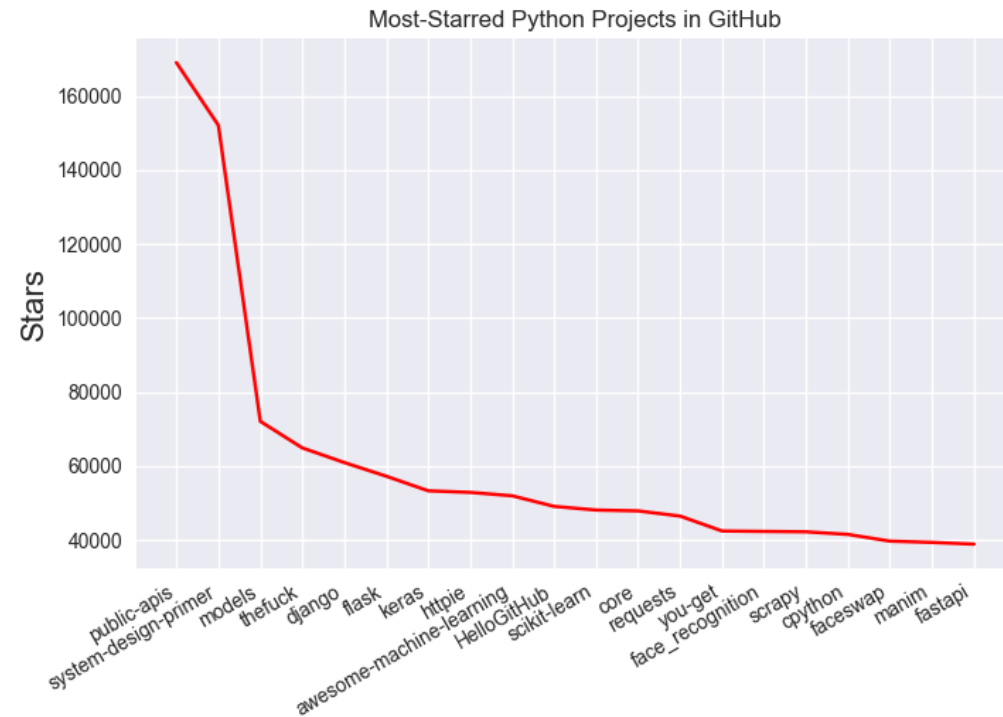
➤ названия осей;

➤ автонаклон меток делений на оси X - *autofmt\_xdate()*.

# ВИЗУАЛИЗАЦИЯ

Визуализация данных:

```
...  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
  
ax.set_title('Most-Starred Python Projects in GitHub')  
ax.set_xlabel('Project', fontsize=16)  
ax.set_ylabel('Stars', fontsize=16)  
fig.autofmt_xdate()  
  
ax.plot(repo_names, stars, c='red')  
  
plt.show()
```



# БЛАГОДАРЮ ЗА ВНИМАНИЕ!

---

Гаврилов Денис Андреевич, преподаватель кафедры СИ ФИТ НГУ