Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Новосибирский национальный исследовательский государственный университет»

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.

ФОРМАТ JSON: ПРИМЕР

```
{
 "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

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 В 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 s**tring»)*

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

```
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'])
```

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},.. ”
```

ДВОЙНАЯ ЗАПИСЬ

```
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'])
```

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.

Корректная запись файла

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

```
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)
```

Построим простой запрос к сервису Github:

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

```
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)
```

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

- Подключение библиотеки *requests*.

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

ПОСТРОЕНИЕ ПРОСТОГО ЗАПРОСА

```
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)
```

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

- 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

```
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)
```

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

- 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

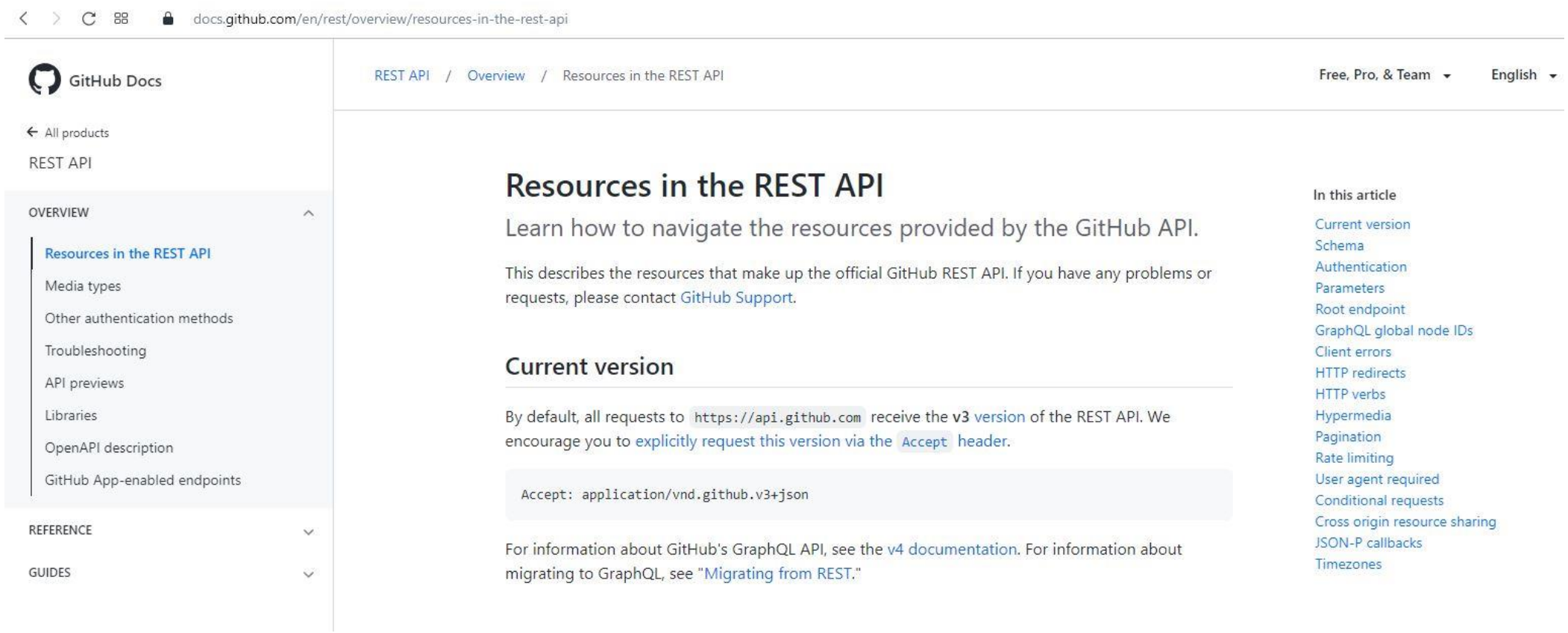
```
import requests

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.

КОНЕЧНЫЕ ТОЧКИ



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}"

ПОСТРОЕНИЕ ЗАПРОСА. АДРЕС

```
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}")
```

Обратимся по адресу, соответствующему конечной точке "repository_search_url".

- Данная точка соответствует результату поиска по репозиториям сервиса Github.
  - *q=language:python&sort=stars* – параметры поиска;
  - Нам интересны данные репозиториев, 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}"

ПОСТРОЕНИЕ ЗАПРОСА. ЗАГОЛОВКИ

```
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}")
```

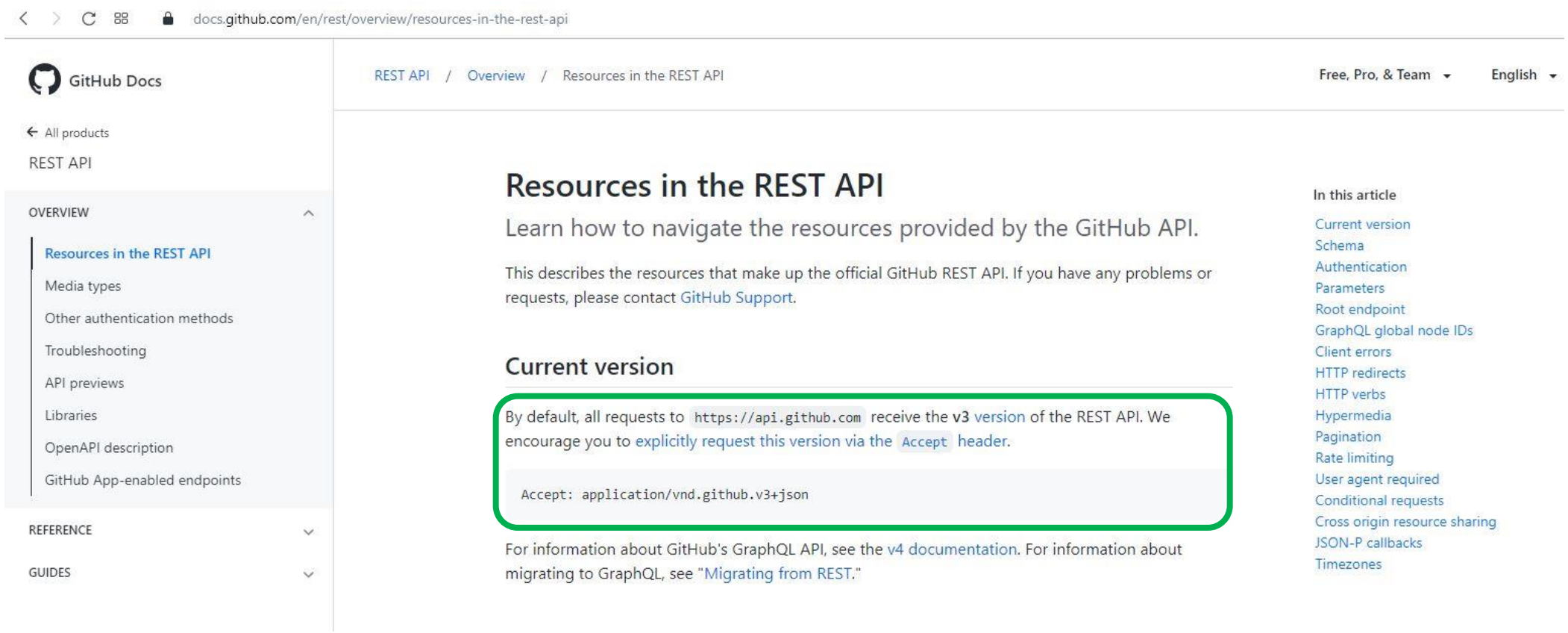
Обратимся по адресу, соответствующему конечной точке "repository_search_url".

- 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}"

ПОСТРОЕНИЕ ЗАПРОСА. ЗАГОЛОВКИ



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

```
. . .
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)}")
```

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

```
# 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

```
. . .
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)}")
```

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

```
# Repositories returned:30
```

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

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}"

ВИЗУАЛИЗАЦИЯ: ПОДГОТОВКА ДАННЫХ

```
. . .
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'])
```

- Сохранение результатов поиска для последующей обработки;
- 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

```
. . .
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()
```

Визуализация данных:

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

ВИЗУАЛИЗАЦИЯ

Визуализация данных:

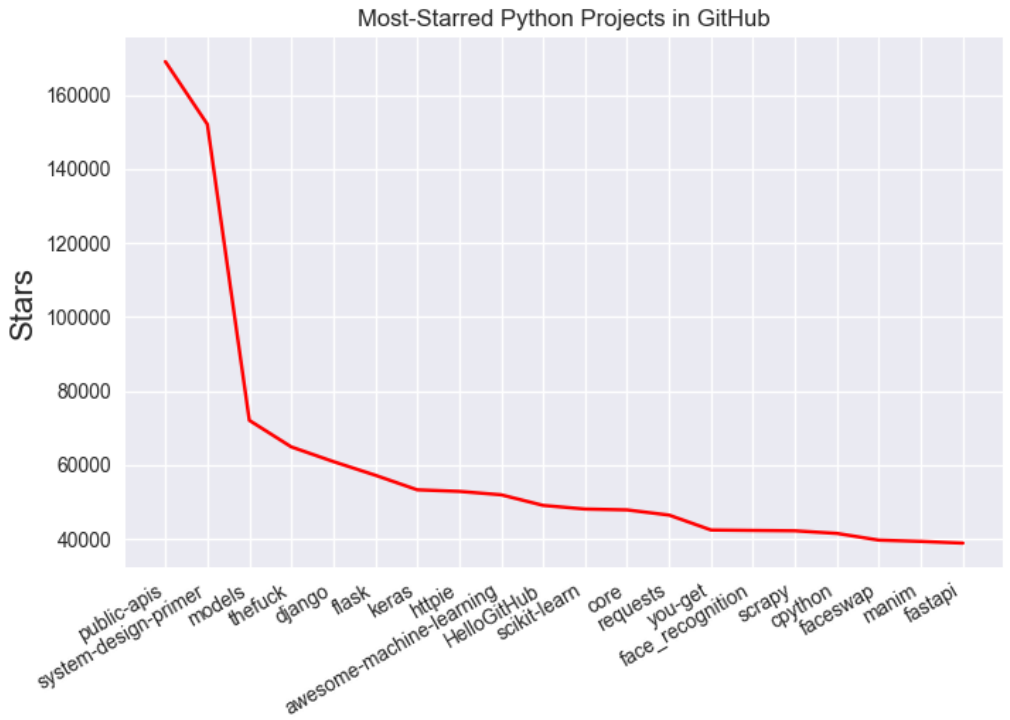
```
. . .
plt.style.use('seaborn')
fig, ax = plt.subplots()

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