Урок WEB. API

Знакомство с API

Аннотация

В уроке рассказывается о том, что такое API, почему и для чего создается API, как изучать новый API и работать с ним. Разбирается HTTP-API на базе StaticMapsAPI и Geocoder API. Изучается принцип работы с HTTP-API на языке Python.
На этом и последующих уроках для работы всех примеров необходимо подключение к сети Интернет.

1. Что такое API

Давайте представим себе, что мы с вами разработали какую-нибудь очень полезную программу. Например, электронную карту города. Программой пользуется все большее количество людей. У нас, конечно же, есть канал обратной связи, по которому пользователи сообщают нам об ошибках в программе и каких функций им не хватает. Мы исправляем ошибки и стараемся расширять функциональность. Но силы наши конечны, а пожеланий с ростом популярности программы становится все больше и больше. Делать все мы не успеваем. Да и нет смысла выполнять каждое пожелание, если мы понимаем, что данная функция нужна лишь очень небольшой аудитории. Но любым отказом мы расстраиваем наших пользователей, а это делать совсем не хочется. Как быть?

Вот одно из решений. Давайте вместо того, чтобы выполнять пожелания пользователей, дадим им инструмент, с помощью которого они сами смогут воплотить свои идеи. Например, можно написать библиотеку (вы ведь знакомы с библиотеками), в которой будут функции, если мы говорим про карты:

- Нарисовать карту
- Нарисовать на ней точки, линии или какие-нибудь картинки

- Построить маршрут по карте
- Найти координаты какого-либо объекта и тому подобное

Понятно, что таким образом мы дадим инструмент только программистам, но программиста можно нанять, и это уже не обязательно должен быть наш сотрудник. Если мы, помимо библиотеки, напишем инструкцию к ней, опишем ее функции, как ими пользоваться и как построить приложение на основе нашей библиотеки, то сторонний программист вполне сможет разобраться с ней и решить задачу заказчика.

И вот так мы как бы строим мостик между нашей **очень полезной программой** и программистом, который хочет пользоваться ее возможностями. Мостик — это интерфейс. В англоязычной терминологии это называется Application Programming Interface или сокращенно API [эй-пи-ай].

API существует у большого количества программных продуктов: от операционной системы до интернет-сервисов. API — это обычный этап развития программных продуктов.

2. Зачем нужен API?

Как правило, API появляется, когда аудитория, использующая программный продукт, разрастается настолько, что своими силами команда-разработчик уже не успевает реализовывать все запросы пользователей.

Естественно, в коммерческом рабочем процессе ничто не делается просто так. Создавая API, разработчики обязательно понимают, какие коммерческие цели они преследуют.

Можно просто предоставлять API за деньги: продавать разовую лицензию или брать плату за обращение к функциям. Часто API — это средство развития платформы. Например, Windows API. Все пишут программы под ОС Windows, в результате чего выигрывает вся платформа. API может решать и репутационные задачи — создавать лояльную пользовательскую аудиторию.

Встречаются и смешанные решения: одновременно существует бесплатное API с определенными ограничениями (использование только в бесплатных открытых продуктах, с ограничением по количеству обращений) и его платная версия без таковых. На этом принципе построено большинство API в Яндексе.

Если это технически реально и логически осмысленно, то в интерфейс можно встраивать рекламу, которая добавляет заработок команде.

Полезно помнить, что API выпускается не просто так, и соотносить свои задачи с используемым инструментом. Особенно важно не забывать про возможные технические (по скорости работы, количеству запросов и т. п.) и юридические (лицензионные) ограничения использования того или иного API.

3. Варианты реализации API

Давайте посмотрим, как может выглядеть API. Один из вариантов — это библиотека, собранная под одну или несколько платформ. Она может либо содержать нужные нам функции или объекты непосредственно в себе, либо использовать Интернет для доступа к удаленному серверу с работающим на нем сервисом. Чтобы начать пользоваться такой библиотекой, надо ее получить (скачать, например), бесплатно или заплатив ее стоимость, установить и дальше применять, как и любую другую библиотеку. Иногда при оплате или регистрации к библиотеке прилагается уникальный ключ для того, чтобы она работала.

Ключ — это хитрая последовательность символов, для которой выполняется какое-то неизвестное нам, но известное авторам ключа условие. Случайно сгенерировать или подобрать ключ очень сложно. Обычно достаточно сложно для того, чтобы проще и дешевле было получить ключ легальным способом.

Есть вариант, при котором скачивать не требуется ничего. Это возможно, если сервис реализован как удаленный, использующий для доступа какой-либо стандартный протокол. Например, HTTP. Обычно для доступа к такому сервису требуется иметь ключ и передавать его при каждом обращении к сервису. Ключ можно получить либо

в автоматическом, либо в полуавтоматическом режиме, как правило, заполнив форму и выразив согласие с лицензионными соглашениями.

Протокол состоит из набора запросов и форматов ответов, предоставляемых сервисом.

HTTP-сервисы можно «пощупать» просто из браузера, поскольку протокол состоит из HTTP-запросов и ответов в форматах, поддерживаемых браузерами.

Пусть для нас изучение HTTP не является самоцелью, но остановиться на нем необходимо. Это тот инструмент, через который мы дальше будем взаимодействовать с различными API.

4. Немного о протоколах взаимодействия

Мы уже много-много раз писали программы, которые работают только на одном компьютере и никак не взаимодействуют с внешним миром. В последние годы приложения, которые функционируют в пределах только одной машины и не смотрят «по сторонам», стали достаточно большой редкостью. Практически в каждую, даже самую маленькую, утилиту автор или авторы старается встроить как минимум механизм автоматического обновления, который поможет быстро доставлять пользователям модули, в которых добавлены новые функции или исправлены ошибки. Самыми массовыми приложениями для взаимодействия в сети Интернет и корпоративных сетях Интранетах являются веб-приложения, от небольших сайтов, до огромных порталов, которые обрабатывают информацию из сотен источников. Но прежде чем мы начнем обращаться к стороннему HTTP API (а затем и создавать свой), сделаем небольшое «лирическое отступление».

Все из вас наверняка знают, что такое компьютерная сеть, но не все задумывались о механизмах взаимодействия устройств внутри сети, ведь даже физическая среда распространения сигналов может быть сильно различна:

- Беспроводная среда (WiFi, 4G, Bluetooth)
- Обычные провода
- Телефонные провода
- Оптические кабели и т. д.

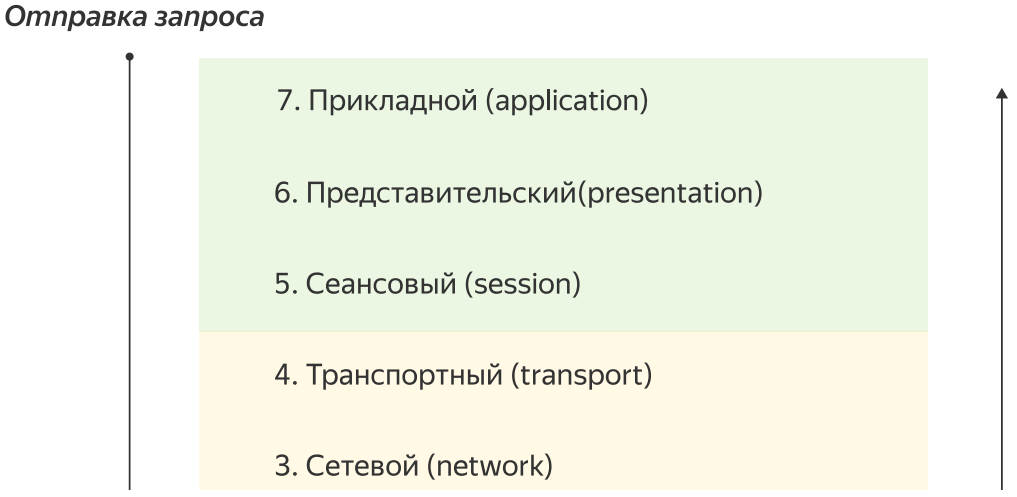
Разумеется, при написании своего приложения большинство программистов не задумывается о том, по каким физическим каналам будет происходить взаимодействие. Это стало возможным благодаря стандартизации в этой области и четкому разделению уровней взаимодействия. Сетевая модель OSI (Open systems interconnection basic reference model) представляет собой набор протоколов (стандартов, описывающих правила взаимодействия разных частей систем при передаче данных), каждый из которых отвечает за определенный уровень взаимодействия. Модель имеет семь уровней:

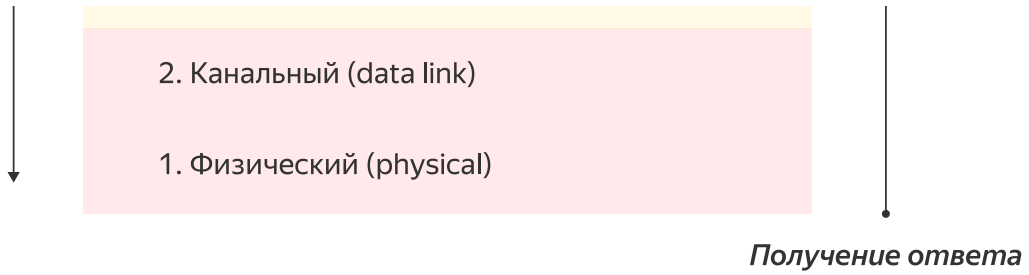
Уровень (layer)	Тип данных	Функции	Примеры
7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3
6. Представительс-кий(presentation)		Представление и шифрование данных	ASCII, EBCDIC
5. Сеансовый (session)		Управление сеансом связи	RPC, PAP
4. Транспортный (transport)	Сегменты / Дейта-граммы	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP, PORTS

3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация	IPsec, IPv4, IPv6, Apple Talk
2. Канальный (data link)	Биты / Кадры	Физическая адресация	PPP, Ethernet, DSL, ARP, IEEE, 802.22
1. Физический (physical)	Биты	Работа со средой передачи, сигналами и двоичными данными	USB, радиоканал, оптоволоконный кабель, «витая пара»

Модель OSI

1. Физический, на котором происходит преобразование двоичных данных в вид, пригодный для передачи в среде (USB, витая пара, WiFi)
2. Канальный, на котором происходит физическая адресация, например, с использованием MAC-адресов сетевой платы
3. Сетевой, на котором происходит логическая адресация. Сюда относятся хорошо известные IPv4 и IPv6
4. Транспортный — для обеспечения связи между конечными точками и обеспечения надежности
5. Сеансовый — для обеспечения сеанса связи
6. Представительский (представления) для представления и шифрования/дешифрования данных.На вышеперечисленные уровни разработчики прикладных программ забираются нечасто, обычно при написании специализированного программного обеспечения и сетевых игр. Протоколы же последнего, седьмого, уровня нужны в повседневной работе гораздо чаще
7. Прикладной, на котором происходит доступ к сетевым службам (SMTP, FTP, HTTP). Именно с использованием протоколов прикладного уровня построен «видимый» интернет, хотя при открытии любой странички по протоколу HTTP ваш запрос проходит по всем уровням модели, превращаясь из данных в физические сигналы, затем собираясь в точке приема обратно для прикладного уровня. Ответ на запрос ждет такая же нелегкая судьба





Более подробно про модель OSI можете почитать, например, на **Википедии**.

Вообще история Интернета довольно увлекательна сама по себе, рекомендуем вам ознакомиться с ней самостоятельно, начать можно, например, на **Википедии**. Мы в нее вдаваться не будем, отметим лишь, что самый первый веб-сайт появился 6 августа 1991 года с доменным именем info.cern.ch. На этом сайте его создатель, Тим Бернерс-Ли, разместил описание новой технологии World Wide Web, основанной на протоколе HTTP, системе адресации URI и языке гипертекстовой разметки HTML.

5. Подробнее о протоколе HTTP

Что же такое HTTP? Это протокол, позволяющий отправлять запросы Интернет-сервису и получать на них ответы.

В протоколе есть несколько разного вида запросов: GET, POST, HEAD, DELETE и т. д.

Как правило, запросы используются следующим образом:

- GET — для получения каких-либо данных с сервера
- POST — для передачи большого объема данных на сервер и получения ответа
- HEAD — для получения информации о данных с сервера
- DELETE — для удаления данных с сервера

Помимо вида запроса обычно требуется указать:

- Адрес сервера, которому запрос направлен
- Путь к конкретной странице на сервере, которую необходимо получить, или к скрипту, который должен обработать запрос
- Дополнительные параметры запроса
- Вид ожидаемого ответа

Протокол HTTP используется браузерами при просмотре интернет-страниц. По умолчанию применяется метод GET, формат ответа — html, json или какой-либо еще стандартный формат, и указывать это при каждом запросе не требуется. Остальная часть запроса записывается в виде:

```
http://static-maps.yandex.ru/1.x/?ll=37.677751,55.757718&spn=0.016457,0.00619&l=map
```

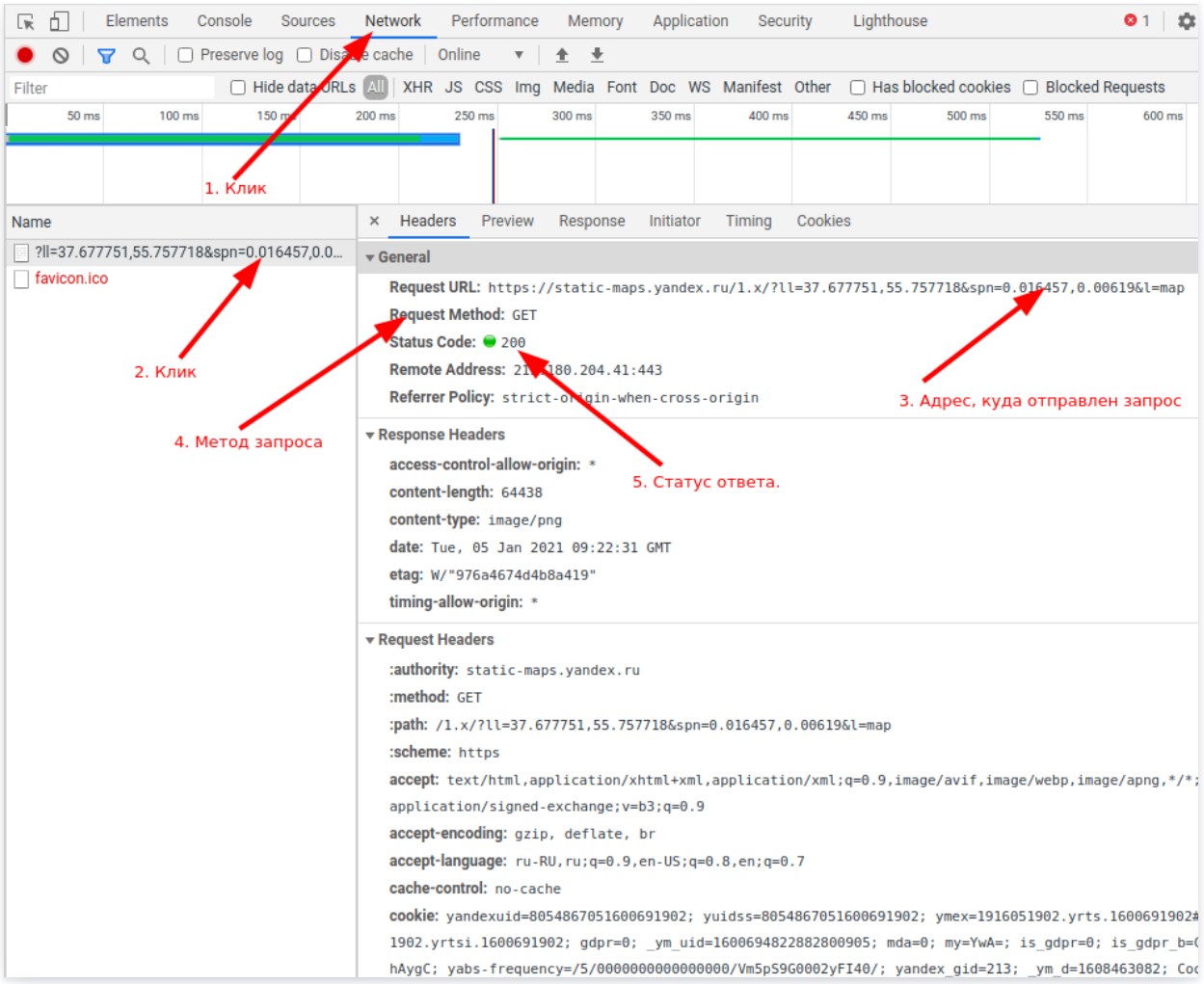
Здесь указаны:

- Название протокола (http)
- Адрес сервера, к которому мы обращаемся (static-maps.yandex.ru)
- Путь к странице на этом сервере (до знака «?») (/1.x/)
- Параметры вида **ключ=значение**, разделенные амперсандом (например, ll=37.677751,55.757718)

Запросы такого вида можно задавать в адресной строке браузера.

Кликните на приведенную ссылку и посмотрите на ответ: **http://static-maps.yandex.ru/1.x/?ll=37.677751,55.757718&spn=0.016457,0.00619&l=map**

В браузере Chrome для этого нажмите клавишу F12.



Ответ состоит из заголовка, в котором указан формат ответа, статус выполнения запроса и сам ответ. Вы наверняка слышали про коды состояния HTTP:

- 200 означает, что все прошло успешно
- 3xx — ошибки, связанные с перемещением страницы и т. п.
- 4xx — ошибки клиента (например, 403 — не хватает прав, 404 — файл не найден)
- 5xx — ошибки уровня приложения (запрос получен, а скрипт, который должен выдать ответ, ответа не выдал)

Если возвращаться к браузеру, то из всего этого пользователю показывается только содержимое ответа в основном окне браузера. При ошибке показывается код и расшифровка ошибки.

Про другие запросы, а также про коды ошибок можно самостоятельно почитать на этой **странице**.

Кстати, среди кодов HTTP-ошибок есть код **418**. Узнайте на досуге, как он расшифровывается и что обозначает, это забавно.

Но давайте от слов перейдем к делу и попробуем пощупать Static API Яндекс.Карт.

6. Знакомство с Yandex.Maps API

Вопрос: откуда нам знать про пути, ключи и их значения, из которых состоят запросы к API? Как понять, в каком формате придет ответ, и что он означает?

На все эти вопросы как раз и отвечает документация API. Страницу с документацией обычно можно обнаружить Поиском. Что же нужно там искать? Давайте посмотрим на содержание этого **ресурса**.

Здесь описываются состав, правила работы и протокол API Yandex.Maps.

HTTP-API состоит из трех частей:

- Static API Карт (получить картинку с картой какой-то области с заданными параметрами)
- Геокодер (поиск топонимов, то есть картографических объектов)
- ППО (Поиск По Организациям)

Кроме того, в документации указано, что ключ для доступа **не нужен**. Лицензионное соглашение предписывает использование API в некоммерческих общедоступных **веб-приложениях**.

Здесь необходимо оговориться, что в рамках данного курса мы рассматриваем серверное программирование. Для того чтобы не нарушать лицензионное соглашение, необходимо, чтобы итоговое приложение имело и веб-составляющую. Понятно, что во время разработки показывать результаты каждого запроса в веб невозможно. Никто этого и не требует. Но в итоговом приложении веб-составляющая должна быть. Пока же мы только изучаем API и не создаем программных продуктов как таковых — это требование нас не затрагивает. Но если в будущем вам предстоит им воспользоваться, учтите это требование сразу и потрудитесь его выполнить. Иначе придется использовать платную версию API.

7. Yandex.Maps Static API

Static API позволяет получить изображение нужного фрагмента карты, которое можно разместить на сайте или в приложении. Такое изображение оптимизировано, «весит» не очень много и загружается быстро даже при медленном Интернете.

Static API возвращает изображение карты в ответ на HTTPS-запрос (HTTPS — защищенная разновидность HTTP). Добавляя в URL разные параметры и задавая их значения, вы можете определить центр карты, ее размер и область показа, отметить нужные объекты и даже отобразить пробки. При этом при каждом новом запросе будет возвращаться изображение с актуальными данными.

Документация для Static API находится на странице:
https://tech.yandex.ru/maps/doc/staticapi/1.x/dg/concepts/input_params-docpage/

Здесь можно почитать про то, какие параметры могут быть в запросе, и что они означают.

Пример запроса: **https://static-maps.yandex.ru/1.x/?ll=37.677751,55.757718&spn=0.016457,0.00619&l=map**

В ответ придет картинка с картой запрошенной области.

Посмотрите, что означают параметры `ll`, `spn`, `l` и поработайте с ними.

Тренировочное задание 1. С помощью запросов к API через браузер получить:

1. Крупномасштабную схему с МГУ им. Ломоносова
2. Спутниковый снимок Эйфелевой башни
3. Спутниковый снимок Авачинского вулкана
4. Спутниковый снимок космодрома Байконур

Подсказка: для решения задачи можно открыть в браузере Я.Карты, найти объекты через поиск, щелк… на карте и получить координаты для параметра `ll`. Параметр `spn` можно подобрать экспериментально.

8. Геокодер (поиск топонимических объектов)

Геокодер помогает определить координаты объекта по его адресу или, наоборот, установить адрес по координатам. К геокодеру можно также обращаться по протоколу HTTPS.

Для обращения к этому API нужен ключ, бесплатный ключ имеет ряд **ограничений**, но он полностью подойдет для наших целей, получить его можно **тут**. Кроме того мы сделали ключ, который может использоваться в рамках нашего курса всеми учащимися проекта, вот он:

apikey = "40d1649f-0493-4b70-98ba-98533de7710b"

Мы предлагаем вам самостоятельно ознакомиться с **документацией**, попробовать сделать запросы, понять, что означают ответы геокодера, и ответить на вопрос второго тренировочного задания.

Тренировочное задание 2. Получите координаты Якутска и Магадана в формате JSON. Выясните, какой город находится севернее: Якутск или Магадан?

9. Обращение к HTTP-сервису на языке Python

Поиграли? Замечательно. Теперь вспомните, что мы программируем на языке Python. Что же нужно для того, чтобы общаться с API Yandex.Карт из программы на Python?

Нужно написать программу, которая выполнит ровно те действия, которые вы только что совершили в браузере руками. Но как?

Нам потребуется библиотека для работы с протоколами http/https. Таких библиотек существует несколько. Все они позволяют передавать запросы и получать ответы от удаленных серверов. Одной из самых популярных библиотек для этого (ее мы и рассмотрим) является библиотека **requests**. Девиз этой библиотеки — HTTP for Humans (HTTP для людей). Она простая в освоении и при этом обладает очень-очень широкой функциональностью.

Библиотека requests не входит в стандартную библиотеку Python, поэтому перед использованием ее надо установить:

```
pip install requests
```

Для выполнения запроса GET (вспоминаем протокол HTTP) используется функция `get()`. Разумеется, предварительно надо импортировать библиотеку :)

```
import requests

response = requests.get("http://geocode-maps.yandex.ru/1.x/?apikey=40d1649f-0493-4b70-98ba-98533de7
print(response, type(response))
```

```
<Response [200]> <class 'requests.models.Response'>
```

Если вместо «красивого» результата вы увидите много-много текста с исключениями, то, скорее всего, вы не смогли подключиться к серверу и получить ответ. В этом случае надо посмотреть на самую последнюю часть ответа и попытаться понять, в чем заключается проблема.

Функция `get()` возвращает объект класса `requests.models.Response`, который среди прочих содержит поля:

- `status_code` — код статуса (200 означает, что запрос выполнен успешно)
- `reason` — текстовая расшифровка статуса на английском языке (например, «Ok» или «Not Found»)
- `content` — ответ сервера

У класса есть метод `_bool_()`, возвращающий True в случае успешного запроса и False в случае ошибки. Если произошла ошибка, то ответ сервера будет пустым. Таким образом, мы можем очень элегантно проверять успешность запроса.

Соберем полученные знания воедино.

Простейшая программа, выполняющая запрос к серверу, анализирующая код ответа и в случае успешного запроса печатающая полученную страницу, выглядит следующим образом:

```
import requests

# Готовим запрос.
geocoder_request = "http://geocode-maps.yandex.ru/1.x/?apikey=40d1649f-0493-4b70-98ba-98533de7710b8

# Выполняем запрос.
response = requests.get(geocoder_request)
if response:
    # Запрос успешно выполнен, печатаем полученные данные.
    print(response.content)
else:
    # Произошла ошибка выполнения запроса. Обрабатываем http-статус.
    print("Ошибка выполнения запроса:")
    print(geocoder_request)
    print("Http статус:", response.status_code, "(", response.reason, ")")
```

```
b'{"response":{"GeoObjectCollection":{"metaDataProperty":{"GeocoderResponseMetaData":{"request":"\
```

Тренировочное задание 3. Попробуйте «испортить» запрос (например, заменив 1.x на 1.x.1), чтобы увидеть, как обрабатывается ошибка запроса.

Если мы передадим в программу строку, содержащую правильно сформулированный запрос к API, то полученный ответ сможем дальше обработать в программе.

Например, из ответа геокодера сможем выбрать и отобразить отдельные поля так, чтобы ответ стал удобочитаемым для пользователя.

Поскольку json — это одним из популярнейших форматов обмена данными, у объекта `requests.Response` уже есть готовый метод `json()`, конструирующий json-объект из текста полученного ответа.

```
import requests

geocoder_request = "http://geocode-maps.yandex.ru/1.x/?apikey=40d1649f-0493-4b70-98ba-98533de7710b8

# Выполняем запрос.
response = requests.get(geocoder_request)
if response:
    # Преобразуем ответ в json-объект
    json_response = response.json()

    # Получаем первый топоним из ответа геокодера.
    # Согласно описанию ответа, он находится по следующему пути:
    toponym = json_response["response"]["GeoObjectCollection"]["featureMember"][0]["Ge
    # Полный адрес топонима:
    toponym_address = toponym["metaDataProperty"]["GeocoderMetaData"]["text"]
```

```
    # Координаты центра топонима:
    toponym_coodrinates = toponym["Point"]["pos"]
    # Печатаем извлечённые из ответа поля:
    print(toponym_address, "имеет координаты:", toponym_coodrinates)
else:
    print("Ошибка выполнения запроса:")
    print(geocoder_request)
    print("Http статус:", response.status_code, "(", response.reason, ")")
```

```
Россия, Республика Саха (Якутия), Якутск имеет координаты: 129.731235 62.027757
```

Запросив изображение карты через Static API, мы можем отрисовать его в окне программы.

Если посмотреть в браузере информацию о странице-картинке, полученной от Static API, то можно найти формат этой картинки. Это PNG. Для того чтобы отобразить картинку на экране, запишем ее в файл с расширением .png и отрисуем файл с помощью библиотеки pygame. Давайте посмотрим, как будет выглядеть такая программа.

```
import os
import sys

import pygame
import requests

map_request = "http://static-maps.yandex.ru/1.x/?ll=37.530887,55.703118&spn=0.002,0.002&l=map"
response = requests.get(map_request)

if not response:
    print("Ошибка выполнения запроса:")
    print(map_request)
    print("Http статус:", response.status_code, "(", response.reason, ")")
    sys.exit(1)

# Запишем полученное изображение в файл.
map_file = "map.png"
with open(map_file, "wb") as file:
    file.write(response.content)

# Инициализируем pygame
pygame.init()
screen = pygame.display.set_mode((600, 450))
# Рисуем картинку, загружаемую из только что созданного файла.
screen.blit(pygame.image.load(map_file), (0, 0))
# Переключаем экран и ждем закрытия окна.
pygame.display.flip()
while pygame.event.wait().type != pygame.QUIT:
    pass
pygame.quit()

# Удаляем за собой файл с изображением.
os.remove(map_file)
```

Нам уже встречалась функция `sys.exit()`, когда мы разрабатывали приложения для PyQt. Она прекр… программы и возвращает результатом значение, переданное в нее в качестве параметра, в блоке по P…

мы возвращали код ошибки Qt, но никто не запрещает нам возвращать информацию, которая поможет нам в отладке программы и выявлении причины завершения приложения.

Разумеется, отображение результатов запроса можно делать не только с помощью PyGame, но и с использованием виджетов PyQt. Давайте перепишем этот же пример:

```
import os
import sys

import requests
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QApplication, QWidget, QLabel

SCREEN_SIZE = [600, 450]


class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.getImage()
        self.initUI()

    def getImage(self):
        map_request = "http://static-maps.yandex.ru/1.x/?ll=37.530887,55.703118&spn=0.002,0.002&l=r
        response = requests.get(map_request)

        if not response:
            print("Ошибка выполнения запроса:")
            print(map_request)
            print("Http статус:", response.status_code, "(", response.reason, ")")
            sys.exit(1)

        # Запишем полученное изображение в файл.
        self.map_file = "map.png"
        with open(self.map_file, "wb") as file:
            file.write(response.content)

    def initUI(self):
        self.setGeometry(100, 100, *SCREEN_SIZE)
        self.setWindowTitle('Отображение карты')

        ## Изображение
        self.pixmap = QPixmap(self.map_file)
        self.image = QLabel(self)
        self.image.move(0, 0)
        self.image.resize(600, 450)
        self.image.setPixmap(self.pixmap)

    def closeEvent(self, event):
        """При закрытии формы подчищаем за собой"""
        os.remove(self.map_file)


if __name__ == '__main__':
    app = QApplication(sys.argv)
```

```
ex = Example()
ex.show()
sys.exit(app.exec())
```

Выполните примеры. Попробуйте задать другие параметры запроса. Подумайте, можно ли обойтись без сохранения полученного контента в файл, а сразу передать его в окно pygame/PyQt?