Практика 5. Работа с API. Часть 1

API (Application Programming Interface) - это программный интерфейс (описание способов взаимодействия одной компьютерной программы с другими, в противоположность пользовательскому интерфейсу, используемому для взаимодействия конечного пользователя с программой)
Более в простой формулировке это список запросов по которым можно обратиться к программе и что она обязуется вернуть в качестве ответа.
Из этого можно сделать вывод, что API - это набор функций программы, которые включают входные и выходные данные, а так же описание того, что делает каждая функция.

REST API (RESTful)

REST (REpresentational State Transfer) - это способ создания API при помощи протокола HTTP.

HTTP

Когда пользователь вводит URL-адрес в браузере на сервер отправляется запрос, такой запрос является GET запросом. После чего сервер формирует и выдаёт ответ. Формат запроса и ответа определяется протоколом HTTP - Hyper Text Transfer Protocol.
В качестве ответа на GET запрос отправляемый с вводом пользователем URL-адреса сервер возвращает данные в формате HTML, который браузер отображает на экране.
Существуют и другие типы запросов, такие как POST, PUT, PATCH, DELETE.

- GET используется для получения со стороны сервера определённого ресурса. По сути это операция чтения.
- POST используется для создания ресурса на сервере. Как правило сервер создаёт в базе данных новую запись и возвращает информацию была ли данная операция успешной.
- PUT и PATCH обновляют информацию на сервере. Обычно это не создаёт новых записей в базе данных, а обновляет существующие. PUT обновляет запись целиком, а PATCH частично.
- DELETE удаляет сущность, как правило запись в базе данных.

Например на сайте есть форма ввода. Когда пользователь нажимает на кнопку "отправить" на сервер отправляется POST запрос.

RESTful веб-сервисы

Для веб-служб, построенных с учётом REST, применяется термин RESTful.
Для RESTful не существует официального стандарта написания API, так как он не является архитектурным стилем.
При разработке RESTful сервиса акцент делается на ресурсе. Например при разработке API для библиотеки в системе одним из ресурсов является книга. Вероятно в базе данных будет существовать одноимённая таблица с набором некоторых полей. Для данной таблицы будет реализован CRUD, на базе которого будет создан набор функций API:

- POST запрос создания книги (Create)
- GET запрос получения книги (Read)
- PATH запрос изменения полей книги (Update)

- DELETE запрос удаления книги (Delete)

Все эти запросы будут под одним общим заголовком /book
По одному адресу может быть несколько типов запросов.
Это может выглядеть следующим образом:

- POST /book для добавления новой книги
- GET /book для получения всех книг
- GET /book/{id} для получения одной книги по id
- DELETE /book/{id} для удаления книги по id

Соответственно при составлении REST API мы определяем какие ресурсы будут доступны для взаимодействия и какие операции с ними будут открыты.

Формат обмена данными

REST не накладывает ограничения на формат обмена данными. Одним из наиболее популярным форматом является JSON.
SOAP (Simple Object Access Protocol), в свою очередь, всегда использует формат XML.

XML

XML (eXtensible Markup Language, расширяемый язык разметки) - язык с простым формальным синтаксисом, удобный для создания и обработки как человеком, так и программой.
Каждый элемент XML должен быть заключён в теги. Название тега заключается в угловые скобки. Тега обычно два - открывающий и закрывающий. В закрывающем теге перед названием добавляется слеш.

```
<Item>
</Item>
```

Тег, с которого начинается и которым заканчивается документ называется корневым элементом. Между открывающим и закрывающим тегом хранится его значение. Кроме обычного текста или числа это могут быть другие теги.

```
<Book>
    <Name>Обрученные холодом</Name>
    <TitleName>Сквозь зеркала</TitleName>
    <Author>Кристель Дабо</Author>
    <ISBN>978-5-00083-674-3</ISBN>
</Book>
```

В данном случае элемент Book имеет четыре параметра Name, TitleName, Author, ISBN с значениями "Обручённые холодом", "Сквозь зеркала", "Кристель Дабо" и "978-5-00083-674-3" соответственно.

Кроме названия и значения у тега могут быть атрибуты.

```
<Book>
    <Name>Обрученные холодом</Name>
    <TitleName>Сквозь зеркала</TitleName>
    <Author>Кристель Дабо</Author>
    <ISBN>978-5-00083-674-3</ISBN>
    <ChaptersList>
        <ChapterItem number=1 page=5>Предисловие</ChapterItem>
        <ChapterItem number=2 page=6>Архивариус</ChapterItem>
        <ChapterItem number=3 page=15>Раскол</ChapterItem>
        <ChapterItem number=4 page=23>Дневник</ChapterItem>
        <ChapterItem number=5 page=42>Медведь</ChapterItem>
        <ChapterItem number=6 page=49>Обсерватория</ChapterItem>
        <ChapterItem number=37 page=335>Проходящая сквозь зеркала</ChapterItem>
    </ChaptersList>
</Book>
```

JSON

JSON (JavaScript Object Notation) - текстовый формат обмена данных основанный на JavaScript. Значением JSON может выступать строка, число (целое или вещественное), логические значения, null, массив и JSON-объект.

JSON объект всегда заключён в фигурные скобки и представляет собой неупорядоченное множество пар ключ:значение. Пример JSON-объекта:

```
{
    "name": "Обрученные холодом",
    "titleName": "Сквозь зеркала",
    "author": "Кристель Дабо",
    "isbn": "978-5-00083-674-3"
}
```

В данном примере все значения являются строками, однако это не обязательно и значения могут быть любые из списка выше, включая другой JSON-объект. Для знакомых с Python легко заметить, что JSON-объект сильно напоминает словарь, по этой причине одно легко конвертируется в другое.

JSON-массив заключается в квадратные скобки.

```
["item1", "item2", "item3", 4, 5, true, false]
```

Элементами массива могут являться все те же самые типы данных, включая другие массивы и JSON-объекты

Расширим данный пример списком глав.

```
{
    "name": "Обрученные холодом",
    "titleName": "Сквозь зеркала",
    "author": "Кристель Дабо",
    "isbn": "978-5-00083-674-3",
    "chapters": [
        {"name": "Предисловие", "number": 1, "page": 5},
        {"name": "Архивариус", "number": 2, "page": 6},
        {"name": "Раскол", "number": 3, "page": 15},
        {"name": "Дневник", "number": 4, "page": 23},
        {"name": "Медведь", "number": 5, "page": 42},
        {"name": "Обсерватория", "number": 6, "page": 49},
        {"name": "Проходящая сквозь зеркала", "number": 37, "page": 335}
    ]
}
```

Практическая часть

Необходимо получить информацию о погоде в своём городе при помощи API OpenWeatherMap (https://openweathermap.org/api).
Для получения ключа необходимо зарегистрироваться (https://home.openweathermap.org/api_keys).

1. Зарегистрируйтесь на OpenWeatherMap и получите ключ (https://home.openweathermap.org/api_keys)

2. Создайте виртуальное окружение Python при помощи `venv`, дальнейшую работу выполняйте в этом виртуальном окружении.

3. Установите библиотеку `requests` в Ваше виртуальное окружение

4. Получите погоду для Вашего города. GET запрос выполняется при помощи `requests.get(url)`, получить помогу можно по адресу `https://api.openweathermap.org/data/2.5/weather?q={city}&lang=ru&units=metric&APPID={key}`
   У объекта ответа будут атрибуты status_code, headers и text

   ```
   import requests

   key = '1234567890'
   city = 'Люберцы'
   url = f'https://api.openweathermap.org/data/2.5/weather?q=
   {city}&lang=ru&units=metric&APPID={key}'

   res = requests.get(url)
   print(res.status_code)
   print(res.headers)
   print(res.text)
   ```

5. Преобразуйте полученную в `.text` строку в формате JSON в объект Python при помощи библиотеки `json`

```
import json

data = json.loads(res.text)
```

6. Выведите на экран информацию о погоде в красивом формате, например:

```
Люберцы
13°C, облачно
Ощущается как 12°C
```