Цифриум

Технологическая карта

Умный код на Python. Базовый уровень

Модуль 3 | Тема 1 | Урок 5

Сохранение данных из API

Уважаемый коллега, обратите внимание!

Учёт присутствия ребёнка на уроке ведётся только по цифровому следу. Каждый ученик из группы на уроке должен передать цифровой след на нашей платформе.

Вы отвечаете за передачу следа учеником на нашей платформе во время урока.

Передавать цифровой след нужно в личном кабинете ученика.

Цифровой след засчитывается по итогу выполнения учеником трёх задач на платформе в ходе урока.

Обращаем внимание — в задачах с отправкой ответа в виде файла наша платформа поддерживает следующие форматы: `.pdf, .doc, .png, .jpeg, .xlsx, .mp3, .py, .ipynb, .txt, .csv, .json, .xml, .sb3, .ino, .hex`.

Контролируйте время урока, чтобы не пропустить момент, когда ученики должны залогиниться в личном кабинете и перейти к решению задач на платформе.

Проследите, чтобы каждый ученик залогинился в личном кабинете и выполнил 3 задачи на платформе. В противном случае, Вам не будет засчитано, что вы провели урок, занятие не будет оплачено, а ученику не будет выставлена отметка о присутствии на уроке.

Просим вас в ходе 45 минут урока проконтролировать, чтобы:

1. Каждый из учеников залогинился в личном кабинете.
2. Каждый ученик в разделе **«Решить задания»** отправил решения по 3 задачам — «Практическое задание 1», «Практическое задание 2» и «Практическое задание 3».

О занятии

Краткое содержание занятия:

На этом занятии мы научимся сохранять данные, полученные из API, в различные форматы файлов. Вы узнаете, как работать с текстовыми файлами, CSV для табличных данных, JSON для структурированной информации и базами данных SQLite для долговременного хранения. Мы создадим программы, которые автоматически собирают данные из API и сохраняют их для дальнейшего анализа.

Цель занятия:

Освоить методы сохранения данных из API в различные форматы для создания автоматизированных систем сбора информации.

Задачи занятия:

- Изучить основные форматы хранения данных.
- Научиться записывать данные в текстовые и CSV-файлы.
- Освоить сохранение и чтение JSON-данных.
- Познакомиться с базами данных SQLite.
- Создать программу для автоматического сбора и сохранения данных из API.
- Понять преимущества и недостатки разных форматов хранения.

Необходимые материалы:

Для проведения занятия понадобятся: рабочая тетрадь, ваш документ/файл для записи терминов и выводов, ваш файл с расширением .py для написания скриптов, доступ к платформе Цифриум для самопроверки и ДЗ.

Темы и время

ЭТАП	ВРЕМЯ
Приветствие	5 мин.
Теория	25 мин.
Итоги занятия	5 мин.
Получение фидбека. Рефлексия. Проверка знаний. Домашнее задание	5 мин.
Вопрос-ответ	5 мин.
Итого	45 мин.

Ход занятия

Номер слайда	Пояснение к слайду
0	**Титульный слайд занятия**
1	**Повторение прошлого занятия**
2	**Анонс текущего занятия**
3	**Постановка цели и задач**
4	Сохранение данных, полученных из API, на локальный диск — это важный практический навык. Во-первых, это позволяет избежать повторных обращений к API для одних и тех же данных, что экономит время и не расходует лимиты на количество запросов, если они есть. Во-вторых, у вас появляется собственная копия данных для анализа, обработки или построения отчётов в офлайн-режиме, вне зависимости от доступности внешнего сервиса. Кроме того, сохранённые данные можно использовать как резервную копию или исторический снимок состояния на определённый момент времени, который потом нельзя будет получить из текущего API. Таким образом, сохранение превращает разовый ответ API в стабильный локальный ресурс для работы.
5	Существует несколько распространённых форматов для сохранения данных, каждый со своей областью применения. Самый простой вариант — текстовые файлы (.txt). Они подходят для сырых или неструктурированных данных, но неудобны для последующего анализа.

Ход занятия

	Для табличных данных лучший выбор — это CSV (Comma-Separated Values). Это простой текстовый формат, который легко читается и Python-библиотеками, и табличными редакторами. Если данные от API уже пришли в формате JSON, логично сохранить их в файл с расширением .json. Это сохранит исходную структуру (словари, списки) и позволит легко загрузить данные обратно в программу. Для более сложных задач с большими объёмами данных или связями между ними используют базы данных, например, лёгкую встраиваемую SQLite. Для отчётов, которые будут просматриваться людьми, удобны Excel-файлы (.xlsx), поддерживающие листы и форматирование.
6	Сохранение данных в простой текстовый файл — это базовый метод. В этом примере мы сначала получаем данные от API и преобразуем ответ в объект Python с помощью .json(). Затем используется конструкция with open(...) as file. Она гарантирует, что файл будет корректно закрыт после записи. Мы открываем (или создаём) файл data.txt в режиме записи ('w') с указанием кодировки UTF-8 для поддержки кириллицы. Поскольку метод .write() ожидает строку, мы преобразуем весь объект data в строковое представление с помощью str(data). Важно понимать, что в таком виде данные теряют свою структуру (словари, списки) и сохраняются в виде одной текстовой строки, что может быть неудобно для последующей программной обработки, но подходит для простого просмотра.
7	CSV (Comma-Separated Values) — это удобный формат для хранения табличных данных. Для работы с ним в Python есть встроенный модуль csv. В примере показано, как создать CSV-файл с нуля. Сначала

Ход занятия

	определяются заголовки столбцов как список headers и данные как список строк rows, где каждая строка — это тоже список значений. Файл открывается в режиме записи 'w'. Ключевые параметры: newline='' предотвращает добавление лишних пустых строк в некоторых операционных системах, а encoding='utf-8' обеспечивает корректную запись кириллицы. Объект csv.writer создаётся для открытого файла. Сначала записывается строка заголовков методом writer.writerow(), затем все строки данных разом с помощью writer.writerows(). Этот формат идеален, если данные от API уже имеют табличную структуру.
8	Для сохранения данных, полученных из API, в их исходном структурированном виде лучшим выбором часто является формат JSON. Для этого используется встроенный модуль json. В примере у нас есть словарь data с вложенной структурой, типичной для ответа API. Файл открывается в режиме записи ('w'). Для записи используется функция json.dump(). Она принимает два основных аргумента: объект Python для сериализации (data) и файловый объект, в который нужно произвести запись (file). Ключевые параметры: ensure_ascii=False позволяет корректно записывать кириллицу и другие не-ASCII символы в читаемом виде, а не в виде escape-последовательностей. indent=4 форматирует выходной файл с отступами, делая его структуру наглядной для человека при просмотре.
9-10	Вопрос для обсуждения

Ход занятия

11	Для более сложного хранения данных, особенно когда нужны связи между таблицами, сложные выборки или частое обновление, используют базы данных. SQLite — отличный вариант для начала. Её главное преимущество в том, что это встраиваемая СУБД. Она не требует отдельного сервера для работы — вся база данных хранится в одном обычном файле на диске (например, database.db). Это делает её портативной и очень простой в настройке. Несмотря на простоту, SQLite поддерживает стандартный язык запросов SQL. Это позволяет выполнять все основные операции: создавать таблицы, вставлять, выбирать, обновлять и удалять данные с помощью знакомых команд. Таким образом, SQLite идеально подходит для прототипирования, небольших проектов, мобильных приложений или любого случая, когда не требуется масштабирование на множество одновременных записей.
12	Первый шаг в работе с SQLite — установить соединение с базой данных и создать объект курсора. Строка sqlite3.connect('api_data.db') создаёт файл базы данных с именем api_data.db (если он не существует) и открывает соединение с ним. Это соединение (conn) управляет всей работой с БД. Затем у соединения вызывается метод .cursor(), чтобы создать объект курсора. Курсор (cursor) — это основной инструмент для выполнения SQL-команд и получения результатов. Именно через него мы будем отправлять запросы к базе данных. Таким образом, эти две строки — это стандартная инициализация, после которой можно приступать к созданию таблиц и работе с данными. Важно помнить, что после завершения всех операций соединение нужно корректно закрыть методом .close().

Ход занятия

13	Прежде чем сохранять данные, необходимо создать структуру для их хранения — таблицу. Для этого используется SQL-команда CREATE TABLE. В примере показана команда создания таблицы users с тремя столбцами: id — целочисленный (INTEGER) первичный ключ (PRIMARY KEY). Он будет автоматически увеличиваться для каждой новой записи, обеспечивая уникальность. name — текстовый тип (TEXT). age — целочисленный тип (INTEGER). Ключевая конструкция IF NOT EXISTS делает команду безопасной: если таблица с именем users уже существует в базе, команда не выполнится и не вызовет ошибки. Это важно для сценариев, которые могут запускаться многократно. Метод cursor.execute() выполняет переданную ему SQL-строку.
14	После создания таблицы в неё можно вставлять данные. Для этого используется SQL-команда INSERT INTO. В примере показана безопасная вставка одной записи. Синтаксис с вопросительными знаками (?, ?) — это параметризованный запрос. Вместо них в кортеже ('Алиса', 25) подставляются фактические значения. Такой подход защищает от SQL-инъекций и корректно экранирует специальные символы. Метод cursor.execute() выполняет команду с переданными параметрами. Критически важный шаг — вызов conn.commit(). По умолчанию SQLite использует транзакции. Это означает, что все изменения (вставка, обновление, удаление) становятся постоянными и записываются в файл базы данных только

Ход занятия

	после явного подтверждения командой commit(). Без этого вызова изменения будут потеряны после закрытия соединения.
15-20	Вопросы для обсуждения

Теория:

Когда мы работаем с API, часто возникает необходимость сохранить полученные данные. Это может быть нужно для создания резервных копий, анализа данных офлайн, соблюдения лимитов API или просто для того, чтобы не делать одни и те же запросы многократно.

Основные форматы хранения данных:

1. Текстовые файлы (.txt)

Самый простой способ — сохранить данные как обычный текст:

```python
import requests

# Получаем данные из API
response =
requests.get('https://api.github.com/users/python')
data = response.json()

# Сохраняем в текстовый файл
with open('github_user.txt', 'w', encoding='utf-8') as
file:
    file.write(f"Имя: {data['name']}\n")
    file.write(f"Компания: {data['company']}\n")
    file.write(f"Репозиториев:
{data['public_repos']}\n")
```

Ход занятия

2. CSV-файлы (.csv)
CSV (Comma-Separated Values) идеально подходит для табличных данных:

Python
```python
import csv
import requests

# Получаем список пользователей
response =
requests.get('https://jsonplaceholder.typicode.com/users
')
users = response.json()

# Сохраняем в CSV
with open('users.csv', 'w', newline='',
encoding='utf-8') as file:
    fieldnames = ['id', 'name', 'email', 'city']
    writer = csv.DictWriter(file, fieldnames=fieldnames)

    writer.writeheader()
    for user in users:
        writer.writerow({
            'id': user['id'],
            'name': user['name'],
            'email': user['email'],
            'city': user['address']['city']
        })
```

3. JSON-файлы (.json)
JSON сохраняет структуру данных и легко читается как человеком, так и программой:

Ход занятия

```python
import json
import requests

# Получаем данные
response =
requests.get('https://jsonplaceholder.typicode.com/posts
?userId=1')
posts = response.json()

# Сохраняем в JSON с форматированием
with open('posts.json', 'w', encoding='utf-8') as file:
    json.dump(posts, file, ensure_ascii=False, indent=2)

# Чтение JSON из файла
with open('posts.json', 'r', encoding='utf-8') as file:
    loaded_posts = json.load(file)
    print(f"Загружено {len(loaded_posts)} постов")
```

4. SQLite база данных
Для больших объёмов данных и сложных запросов лучше использовать базу данных:

```python
import sqlite3
import requests

# Создаем подключение к БД
conn = sqlite3.connect('api_data.db')
cursor = conn.cursor()

# Создаем таблицу
cursor.execute('''
```

Ход занятия

```
    CREATE TABLE IF NOT EXISTS posts (
        id INTEGER PRIMARY KEY,
        user_id INTEGER,
        title TEXT,
        body TEXT
    )
''')

# Получаем данные из API
response =
requests.get('https://jsonplaceholder.typicode.com/posts
')
posts = response.json()

# Сохраняем в БД
for post in posts:
    cursor.execute('''
        INSERT OR REPLACE INTO posts (id, user_id,
title, body)
        VALUES (?, ?, ?, ?)
    ''', (post['id'], post['userId'], post['title'],
post['body']))

conn.commit()
conn.close()
```

Обработка ошибок при сохранении:

Python

```
import requests
import json
from datetime import datetime
```

Ход занятия

```
def save_api_data(url, filename):
    try:
        response = requests.get(url, timeout=10)
        response.raise_for_status()

        data = response.json()

        # Добавляем метаданные
        result = {
            'timestamp': datetime.now().isoformat(),
            'source': url,
            'data': data
        }

        with open(filename, 'w', encoding='utf-8') as
file:
            json.dump(result, file, ensure_ascii=False,
indent=2)

        print(f"Данные успешно сохранены в {filename}")
        return True

    except requests.exceptions.RequestException as e:
        print(f"Ошибка при запросе: {e}")
        return False
    except json.JSONDecodeError as e:
        print(f"Ошибка при разборе JSON: {e}")
        return False
    except IOError as e:
        print(f"Ошибка при записи файла: {e}")
        return False
```

Автоматизация сбора данных:

Ход занятия

```python
import time
import schedule

def collect_weather_data():
    # Здесь код для сбора данных о погоде
    print("Собираю данные о погоде...")
    # save_api_data(weather_api_url,
'weather_data.json')

# Запускать каждый час
schedule.every().hour.do(collect_weather_data)

# Бесконечный цикл
while True:
    schedule.run_pending()
    time.sleep(60)
```

Выбор формата хранения:

- TXT: для простых логов и отчётов;
- CSV: для табличных данных, работы в Excel;
- JSON: для сохранения структуры данных;
- SQLite: для больших объёмов и сложных запросов.

Практика

Вопросы для самопроверки

1. Какой формат файла лучше выбрать для сохранения списка пользователей с их контактными данными, если планируется анализ в Excel?
2. Почему при работе с SQLite важно использовать параметризованные запросы вместо форматирования строк?
3. В чем преимущество сохранения данных из API в локальные файлы?

Практическое задание

Задача 1:

Создайте программу, которая получает информацию о курсах валют с API Центробанка (или любого другого открытого API) и сохраняет данные в CSV-файл. Файл должен содержать столбцы: код валюты, название, курс к рублю, дата. Добавьте обработку ошибок на случай недоступности API.

Задача 2:

Напишите скрипт, который собирает данные о репозиториях пользователя GitHub через API и сохраняет их в JSON-файл со следующей структурой:

- Информация о пользователе (имя, компания, количество репозиториев).
- Список репозиториев с полями: название, описание, язык программирования, количество звёзд.
- Дата и время сбора данных.

Задача 3:

Создайте систему мониторинга погоды с использованием SQLite. Программа должна:

- Каждые 30 минут получать данные о погоде через API
- Сохранять в базу данных: температуру, влажность, давление, время замера
- Реализовать функцию вывода средней температуры за день
- Добавить возможность экспорта данных за определенный период в CSV

Проверь себя по следующим критериям:

Практика

- Программа корректно обрабатывает ответы API и извлекает нужные данные.
- Реализована обработка ошибок (недоступность API, некорректные данные).
- Данные сохраняются в указанном формате с правильной структурой.
- Для работы с файлами используются контекстные менеджеры (with).
- В SQLite используются параметризованные запросы для безопасности.

Примеры выполнения заданий

Задача 1:

Python

```
import csv
import requests
from datetime import datetime

def get_exchange_rates():
    """Получает курсы валют с API"""
    # Пример с использованием exchangerate-api.com
    url =
"https://api.exchangerate-api.com/v4/latest/RUB"

    try:
        response = requests.get(url, timeout=10)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Ошибка при получении данных: {e}")
        return None

def save_to_csv(data):
    """Сохраняет курсы валют в CSV файл"""
    if not data:
        return

    filename =
f"exchange_rates_{datetime.now().strftime('%Y%m%d')}.csv
"

    with open(filename, 'w', newline='',
encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow(['Код валюты', 'Курс к рублю',
'Дата'])
```

Примеры выполнения заданий

```
        date = data.get('date',
datetime.now().strftime('%Y-%m-%d'))
        rates = data.get('rates', {})

        for currency, rate in rates.items():
            if currency != 'RUB':  # Пропускаем рубль к
рублю
                # Конвертируем: если 1 RUB = rate USD,
то 1 USD = 1/rate RUB
                rub_rate = 1 / rate if rate != 0 else 0
                writer.writerow([currency,
round(rub_rate, 4), date])

    print(f"Данные сохранены в {filename}")

# Основная программа
if __name__ == "__main__":
    rates_data = get_exchange_rates()
    save_to_csv(rates_data)
```

Задача 2:

Python

```
import json
import requests
from datetime import datetime

def get_github_user_data(username):
    """Получает данные пользователя и его
репозиториев"""
    base_url = "https://api.github.com"

    result = {
```

Примеры выполнения заданий

```
        'timestamp': datetime.now().isoformat(),
        'user_info': {},
        'repositories': []
    }

    try:
        # Получаем информацию о пользователе
        user_response =
requests.get(f"{base_url}/users/{username}")
        user_response.raise_for_status()
        user_data = user_response.json()

        result['user_info'] = {
            'name': user_data.get('name', 'N/A'),
            'company': user_data.get('company', 'N/A'),
            'public_repos':
user_data.get('public_repos', 0),
            'followers': user_data.get('followers', 0)
        }

        # Получаем репозитории
        repos_response =
requests.get(f"{base_url}/users/{username}/repos")
        repos_response.raise_for_status()
        repos_data = repos_response.json()

        for repo in repos_data:
            result['repositories'].append({
                'name': repo['name'],
                'description': repo['description'] or
'Без описания',
                'language': repo['language'] or 'Не
указан',
                'stars': repo['stargazers_count'],
                'forks': repo['forks_count'],
                'created_at': repo['created_at']
```

Примеры выполнения заданий

```
            })

        # Сохраняем в JSON
        filename =
f"github_{username}_{datetime.now().strftime('%Y%m%d_%H%
M%S')}.json"
        with open(filename, 'w', encoding='utf-8') as
file:
            json.dump(result, file, ensure_ascii=False,
indent=2)

        print(f"Данные сохранены в {filename}")
        print(f"Найдено репозиториев:
{len(result['repositories'])}")

    except requests.exceptions.RequestException as e:
        print(f"Ошибка при работе с API: {e}")

# Использование
if __name__ == "__main__":
    username = input("Введите имя пользователя GitHub:
")
    get_github_user_data(username)
```

Задача 3:

Python

```
import sqlite3
import requests
import time
from datetime import datetime, timedelta
```

Примеры выполнения заданий

```
class WeatherMonitor:
    def __init__(self, db_name='weather.db'):
        self.conn = sqlite3.connect(db_name)
        self.cursor = self.conn.cursor()
        self.create_table()

    def create_table(self):
        """Создает таблицу для хранения данных о
погоде"""
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS weather_data (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                timestamp DATETIME DEFAULT
CURRENT_TIMESTAMP,
                temperature REAL,
                humidity INTEGER,
                pressure REAL,
                description TEXT
            )
        ''')
        self.conn.commit()

    def fetch_weather(self, city='Moscow'):
        """Получает данные о погоде через API"""
        # Используйте свой API ключ от OpenWeatherMap
        api_key = "YOUR_API_KEY"
        url =
f"http://api.openweathermap.org/data/2.5/weather?q={city
}&appid={api_key}&units=metric"

        try:
            response = requests.get(url)
            response.raise_for_status()
            data = response.json()

            return {
```

Примеры выполнения заданий

```
                'temperature': data['main']['temp'],
                'humidity': data['main']['humidity'],
                'pressure': data['main']['pressure'],
                'description':
data['weather'][0]['description']
            }
        except Exception as e:
            print(f"Ошибка при получении погоды: {e}")
            return None

    def save_weather(self, weather_data):
        """Сохраняет данные о погоде в БД"""
        if weather_data:
            self.cursor.execute('''
                INSERT INTO weather_data (temperature,
humidity, pressure, description)
                VALUES (?, ?, ?, ?)
            ''', (
                weather_data['temperature'],
                weather_data['humidity'],
                weather_data['pressure'],
                weather_data['description']
            ))
            self.conn.commit()
            print(f"Данные сохранены:
{weather_data['temperature']}°C")

    def get_daily_average(self, date=None):
        """Возвращает среднюю температуру за день"""
        if date is None:
            date = datetime.now().date()

        query = '''
            SELECT AVG(temperature)
            FROM weather_data
            WHERE DATE(timestamp) = ?
```

Примеры выполнения заданий

```
        '''

        self.cursor.execute(query, (date,))
        result = self.cursor.fetchone()
        return result[0] if result[0] else 0

    def export_to_csv(self, start_date, end_date,
filename='weather_export.csv'):
        """Экспортирует данные за период в CSV"""
        import csv

        query = '''
            SELECT timestamp, temperature, humidity,
pressure, description
            FROM weather_data
            WHERE timestamp BETWEEN ? AND ?
            ORDER BY timestamp
        '''

        self.cursor.execute(query, (start_date,
end_date))
        data = self.cursor.fetchall()

        with open(filename, 'w', newline='',
encoding='utf-8') as file:
            writer = csv.writer(file)
            writer.writerow(['Время', 'Температура',
'Влажность', 'Давление', 'Описание'])
            writer.writerows(data)

        print(f"Экспортировано {len(data)} записей в
{filename}")

    def monitor_loop(self, interval_minutes=30):
        """Основной цикл мониторинга"""
```

Примеры выполнения заданий

```
        print(f"Начинаю мониторинг погоды каждые
{interval_minutes} минут...")

        while True:
            weather = self.fetch_weather()
            self.save_weather(weather)

            # Показываем среднюю температуру за сегодня
            avg_temp = self.get_daily_average()
            print(f"Средняя температура сегодня:
{avg_temp:.1f}°C")

            # Ждем следующего измерения
            time.sleep(interval_minutes * 60)

    def close(self):
        """Закрывает соединение с БД"""
        self.conn.close()

# Пример использования
if __name__ == "__main__":
    monitor = WeatherMonitor()

    # Для демонстрации добавим тестовые данные
    test_data = {
        'temperature': 20.5,
        'humidity': 65,
        'pressure': 1013.25,
        'description': 'облачно'
    }
    monitor.save_weather(test_data)

    # Получаем среднюю температуру
    avg = monitor.get_daily_average()
    print(f"Средняя температура сегодня: {avg:.1f}°C")
```

Примеры выполнения заданий

```
# Экспорт данных за последние 7 дней
end_date = datetime.now()
start_date = end_date - timedelta(days=7)
monitor.export_to_csv(start_date, end_date)

# Для реального мониторинга раскомментируйте:
# monitor.monitor_loop(30)

monitor.close()
```

Сегодня на занятии вы:

- Изучили способы сохранения данных из API.
- Освоили работу с CSV и JSON файлами.
- Познакомились с SQLite.
- Научились выбирать подходящий формат.
- Создали программы для автоматизации сбора данных.

На следующем занятии вы:

- Познакомитесь с HTML-структурой веб-страниц.
- Изучите библиотеку BeautifulSoup.
- Научитесь извлекать данные из HTML.
- Создадите свой первый парсер.

Получение фидбека. Рефлексия. Проверка знаний

Попросите ребят решить интерактивные задачи по теме урока.

Ответы к интерактивным задачам

Практическое задание 1

Какой формат хранения данных лучше всего подходит для табличных данных и работы в Excel?

А) JSON

Б) TXT

В) CSV

Г) XML

Практическое задание 2

Почему важно использовать параметризованные запросы (?, ?) при работе с SQLite?

А) Для ускорения выполнения запросов

Б) Для красивого форматирования кода

В) Для защиты от SQL-инъекций

Г) Для поддержки кириллицы

Практическое задание 3

Какой метод используется для подтверждения изменений в базе данных SQLite после выполнения запросов?

commit

Ответы к интерактивным задачам

Вопрос-ответ

Если во время подготовки или проведения урока возникают вопросы, их необходимо адресовать в методический чат.