Цифриум

Технологическая карта

Умный код на Python. Базовый уровень

Модуль 3 | Тема 1 | Урок 3

Простые публичные API (часть 2)

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:

Вы узнаете, как получать данные из публичных API постранично, разбирать вложенные JSON-структуры, обрабатывать ошибки при запросах и учитывать ограничения API.

Цель занятия:

Закрепить навыки работы с публичными API для эффективного получения и обработки данных с учетом особенностей использования API.

Задачи занятия:

1. Разобраться, что такое API и как оно работает.
2. Изучить понятие пагинации и научиться получать данные частями (постранично) из API.
3. Освоить разбор и извлечение информации из вложенных JSON-структур.
4. Познакомиться с основами обработки ошибок при запросах к API и написанием устойчивого к ошибкам кода на Python.
5. Узнать про ограничения публичных API: лимиты запросов, частоту вызовов и авторизацию.

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.

Термины:

Пагинация — это способ получения данных частями, когда их слишком много для одного ответа.
Ручная пагинация — самостоятельное разбиение большого объёма данных на части, позволяющая работать с данными по страницам.
Точечная нотация — прямой доступ к вложенным элементам через ключи: data["user"]["address"]["city"].
Понимание списков — удобный способ пройтись по спискам и извлечь нужные данные с фильтрацией.
Рекурсия — динамический обход вложенных структур с помощью функции, принимающей цепочку ключей.

О занятии

Обработка отсутствующих ключей — методы безопасного доступа, например dict.get(), или наличие ключа перед обращением;обращением.
Преобразование JSON-строк в объекты Python — с помощью модуля json (json.loads()), чтобы работать с данными как со словарями и списками.
Использование библиотек или методов для удобного доступа - например, pydash.get(), jsonpath, упрощающие выборку из сложных структур.

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	Пагинация — это техника разделения большого набора данных на отдельные страницы (порции) при работе с API. Когда серверу нужно вернуть, например, 10 000 записей, он не отправляет их все сразу одним ответом. Это делается по нескольким причинам. Во-первых, для производительности: маленькие порции данных быстрее генерируются сервером и обрабатываются клиентом. Во-вторых, это экономит ресурсы как сервера, так и клиентского приложения, предотвращая перегрузку памяти. Для пользователя это тоже удобно — интерфейс может подгружать данные постепенно, по мере необходимости. Кроме того, пагинация делает систему более устойчивой: при ошибке загрузки одной страницы не нужно перезагружать весь массив данных, а только проблемную часть. В API пагинация обычно реализуется через параметры запроса, такие как page и limit.
5	На практике пагинация работает так: клиент в запросе указывает, какую именно порцию данных он хочет получить. Сервер возвращает только эту порцию, а также часто сообщает общую информацию — например, общее количество страниц или записей. Клиент управляет загрузкой, изменяя параметры запроса. Самые распространённые параметры — это page (номер

Ход занятия

	страницы) и limit или per_page (количество элементов на странице). Например, в запросе https://api.example.com/items?page=5&limit=20 клиент просит пятую страницу, где каждая страница содержит по 20 элементов. Таким образом, он получит элементы с 81-го по 100-й. Это позволяет API оставаться отзывчивым даже при работе с огромными наборами данных, а клиенту — гибко управлять загрузкой информации.
6	Давайте разберём конкретный числовой пример. Предположим, у сервера есть 100 элементов, которые нужно вернуть. Если клиент установил limit=20, то сервер разбивает всю выборку на страницы, где на каждой помещается 20 элементов. Получается 5 страниц. Нумерация обычно начинается с 1. Таким образом: Страница 1 содержит элементы с 1-го по 20-й. Страница 2 — с 21-го по 40-й. Страница 3 — с 41-го по 60-й. Страница 4 — с 61-го по 80-й. Страница 5 — с 81-го по 100-й. Запрос ?page=5&limit=20 означает: «Дай мне пятую страницу, где размер страницы — 20 элементов». В ответ сервер отправит последние 20 элементов, с 81-го по 100-й. Это и есть пагинация в действии.
7	Бывают ситуации, когда API не реализует пагинацию на своей стороне и возвращает все данные одним большим списком. В таком случае ответственность за разбиение на страницы ложится на клиента — это называется ручной (или

Ход занятия

	клиентской) пагинацией. Алгоритм такой: мы делаем один запрос и получаем весь массив данных, например, список из 100 пород собак. Затем уже в нашем коде на Python мы разбиваем этот список на части нужного размера, например, по 5 элементов на страницу, и показываем пользователю по одной части за раз. В следующем примере мы сделаем именно это: запросим полный список из API, а затем с помощью срезов списка и цикла организуем постраничный вывод, имитируя работу с пагинацией на стороне клиента.
8	В этом примере показана реализация ручной пагинации. Мы используем API, который возвращает полный словарь всех пород собак одним запросом. Алгоритм работы кода: сначала мы отправляем запрос и извлекаем из ответа весь список пород, преобразуя его в список Python. Затем определяем логику пагинации: переменная limit задаёт размер страницы (5 пород), а page — текущий номер страницы. Формулы start = (page - 1) * limit и end = page * limit вычисляют, какой срез списка соответствует нужной странице. Например, для страницы 1 это элементы с 0 по 4. Мы извлекаем этот срез, проверяем, не пустой ли он (это будет сигналом окончания данных), и выводим породы на текущей странице. После этого увеличиваем номер страницы и повторяем процесс для следующего среза, пока не дойдём до конца списка. Таким образом, мы имитируем постраничный вывод на стороне клиента.
9	Что такое JSON? Это текстовый формат обмена данными, основанный на синтаксисе JavaScript, но независимый от языка. Он представляет структуры данных в виде пар «ключ-значение» и упорядоченных списков, что делает его идеальным для передачи данных между сервером и клиентом.

Ход занятия

	Какие типы данных могут быть в JSON? Допустимые типы: строки, числа, логические значения (true/false), значение null, объекты (словари в Python) и массивы (списки в Python). Как получить элемент из списка в JSON? Когда JSON-ответ преобразован в структуру Python (например, с помощью response.json()), вы работаете со стандартными типами. Чтобы получить элемент списка, используйте индексацию: data_list[0] для первого элемента. Как работать с вложенными структурами? Доступ к данным осуществляется через цепочку ключей и индексов. Например, для доступа к значению внутри словаря, который находится в списке внутри другого словаря, используется конструкция вида: data['ключ_словаря'][индекс_списка]['вложенный_ключ'].
10	Перед нами пример JSON-объекта с вложенной структурой. Это типичный ответ от API, содержащий информацию о пользователе. Внешний объект имеет один ключ — "user". Его значением является другой объект (словарь). Внутри этого вложенного объекта мы видим разные типы данных: простые пары «ключ: значение» ("id": 123, "name": "Анна"), ещё один вложенный объект "contacts", а также массивы (списки) "phones" и "roles". Таким образом, данные организованы иерархически. Чтобы извлечь, например, email, нужно пройти по цепочке ключей: сначала обратиться к "user", затем к "contacts", и наконец к "email". Списки позволяют хранить множественные значения, например, несколько телефонных номеров или ролей пользователя в системе.
11	Для работы с такими сложными JSON-структурами есть несколько основных методик. Основной способ — последовательный доступ по ключам и

Ход занятия

	индексам, используя квадратные скобки: data["user"]["contacts"]["email"]. Это прямая навигация по известной структуре. Для безопасного доступа, когда ключ может отсутствовать, используйте метод .get(): data.get("user", {}).get("email"). Это предотвратит ошибку KeyError. Если нужно обработать все элементы в списке внутри JSON (например, все номера телефонов), применяется цикл или list comprehension: [phone for phone in data["user"]["contacts"]["phones"]]. Для разбора очень сложных или неизвестных заранее структур могут применяться более продвинутые методы: рекурсивный обход или специализированные библиотеки вроде jsonpath, которые позволяют извлекать данные по шаблону пути, аналогично XPath для XML. Но в большинстве случаев достаточно прямого доступа по ключам и метода .get().
12	В этом примере показана практическая работа с вложенной структурой от API собак. Сначала мы получаем ответ и преобразуем его в словарь Python. Затем с помощью метода .get() безопасно извлекаем основной словарь пород по ключу "message". Если словарь пуст, выводим сообщение об этом. Далее получаем первую породу из словаря, преобразовав его ключи в список. Для этой породы снова используем .get(), чтобы безопасно получить список её подвидов. Если список не пуст, выводим подвиды, иначе — сообщение об их отсутствии. Этот подход демонстрирует два ключевых принципа: прямой доступ по цепочке ключей (data["message"]) и безопасный доступ с проверкой и значением по умолчанию через .get(), что предотвращает ошибки при отсутствии ожидаемых ключей в ответе API.

Ход занятия

13	При работе с данными от API придерживайтесь нескольких практических правил. Первым делом всегда визуализируйте полную структуру полученного JSON, используя json.dumps() с параметром indent. Это даёт чёткое понимание того, как организованы данные. При написании кода выделяйте повторяющиеся операции по извлечению данных в отдельные функции — это сделает код чище и упростит его поддержку. Крайне важно обрабатывать случаи отсутствия ожидаемых ключей. Используйте метод .get() с значением по умолчанию или конструкции try-except, чтобы программа не завершалась аварийно из-за KeyError. Делайте код читаемым: сохраняйте промежуточные части сложного пути в переменные с понятными именами, например, user_contacts = data["user"]["contacts"]. Если структура данных очень сложная и глубокая, рассмотрите использование специализированных инструментов, таких как библиотека jsonpath, для более элегантного извлечения данных по шаблонам.
14	Работа с API всегда должна включать обработку ошибок, так как вы зависите от внешнего сервиса. Ошибки можно разделить на несколько категорий. Первая — это HTTP-ошибки, когда сервер возвращает код состояния 4xx или 5xx. Например, 404, если ресурс не найден, или 500 при внутренней ошибке сервера. Вторая категория — сетевые проблемы: отсутствие интернет-соединения, таймаут запроса или отказ DNS. В таких случаях библиотека requests выбросит соответствующее исключение, например, ConnectionError. Третья группа — ошибки парсинга ответа. Это происходит, если сервер вернул данные в неожиданном формате, не в

Ход занятия

	JSON, и вызов response.json() завершится с ошибкой JSONDecodeError. И, наконец, логические ошибки: API отвечает кодом 200 (успех), но возвращает пустой массив, некорректные или неполные данные. Такие ошибки нужно обрабатывать проверкой содержимого ответа.
15	Обработка ошибок — обязательная часть работы с любым внешним API. Без неё ваше приложение будет нестабильным. Ключевой инструмент для этого в Python — блок try/except. В нём вы оборачиваете код, который может вызвать ошибку: сам сетевой запрос, парсинг JSON или проверку данных. Для выявления стандартных HTTP-ошибок (коды 4xx, 5xx) используйте метод response.raise_for_status(). Если статус ответа указывает на ошибку, этот метод выбросит исключение requests.HTTPError, которое можно перехватить и обработать. Таким образом, ваш код должен последовательно проверять и обрабатывать разные уровни проблем: сначала сетевые сбои и таймауты, затем HTTP-статусы ответа, и только после успешных проверок — парсить и анализировать сами данные. Это делает программу устойчивой к сбоям во внешнем сервисе.
16	В этом примере показан базовый, но правильный подход к обработке ошибок при запросе к API. Весь код, связанный с внешним вызовом, помещён в блок try. Первым делом выполняется сам запрос с помощью requests.get(). Следующий критически важный шаг — вызов response.raise_for_status(). Этот метод проверяет код статуса HTTP в ответе. Если статус указывает на ошибку (например, 404 или 500), метод генерирует исключение HTTPError, которое прервёт выполнение и перейдёт в блок except.

Ход занятия

	Только если статус успешный (2xx), выполнение продолжается, и мы пытаемся преобразовать ответ в JSON с помощью response.json(). Этот шаг также потенциально может вызвать ошибку, если ответ сервера не является валидным JSON, что потребует дополнительной обработки в блоке except.
17	В этой части примера показаны блоки except, которые перехватывают и обрабатывают различные типы ошибок. Каждый блок except ловит конкретное исключение, что позволяет дать пользователю точное и понятное сообщение о проблеме: HTTPError — перехватывает ошибки, сгенерированные методом raise_for_status(), например, «404 Client Error». ConnectionError — указывает на проблемы с установкой соединения (нет сети, неверный адрес). Timeout — возникает, когда сервер не отвечает в заданный промежуток времени. RequestException — это базовое исключение библиотеки requests для всех остальных ошибок, связанных с запросом. ValueError — ловит ошибку, которая может возникнуть при вызове response.json(), если тело ответа не является валидным JSON. Такой структурированный подход обеспечивает надёжность: программа не «упадёт», а корректно сообщит о типе возникшей проблемы.
18-19	При использовании любого публичного API важно учитывать его ограничения. Это правила, которые сервис устанавливает для поддержания стабильности и безопасности. Самые частые ограничения — это лимит запросов (rate limiting), который не позволяет отправлять больше N

Ход занятия

	запросов в секунду, минуту или день. Также часто встречаются ограничения на объём возвращаемых данных, например, не более 100 записей на один запрос. Многие API требуют авторизации через ключ или токен. Без него запросы будут отклоняться. Кроме того, могут быть территориальные ограничения или ограничения по времени доступа. Также сервис может разрешать только определённые типы запросов (например, только GET) и возвращать данные в строго заданном формате, без возможности кастомизации ответа. Все эти условия обычно описаны в документации разработчика (Developer Docs).

Практика

Вопросы для самопроверки

1. Что такое пагинация при работе с API?
2. Какие параметры запроса позволяют управлять пагинацией?
3. Чем полезен метод try/except?
4. Какие типы ошибок могут возникать при работе с API?
5. Какие ограничения могут быть у публичных API?

Практическое задание

Задача 1: Получите полный список пород собак с API https://dog.ceo/api/breeds/list/all и разбейте его на страницы по 5 пород. Выведите первые 3 страницы.

Задача 2: Используя пример вложенного JSON с пользователем, напишите код, который выведет email пользователя и первый номер телефона.

Пример вложенного JSON с пользователем:

```
{
  "user": {
    "id": 123,
    "name": "Анна",
    "contacts": {
      "email": "anna@example.com",
      "phones": ["+123456789", "+987654321"]
    },
    "roles": ["admin", "editor"]
  }}
```

Задача 3: Напишите функцию, которая безопасно извлекает поле "address", "city" из JSON, возвращая "Не указано", если такого поля нет.

Практика

Задача 4: Сделайте запрос к любому публичному API и обработайте возможные ошибки: HTTP ошибки, таймаут, некорректный JSON. В случае ошибки выведите сообщение для пользователя.

Задача 5: Напишите запрос к API к https://dog.ceo/api/breeds/list/all, получите JSON-ответ и рекурсивно найдите в нем список пород собак по цепочке ключей.

Проверь себя по следующим критериям:

- Код выполняется без ошибок
- Ответы выведены в консоль
- Правильно обработаны JSON-данные
- API-запросы выполняются корректно
- Реализована обработка ошибок

Примеры выполнения заданий

Ответы на вопросы самопроверки

1. Что такое пагинация при работе с API?

Пагинация — это техника разбиения большого набора данных на отдельные страницы (порции). Это позволяет не загружать все данные сразу, что повышает производительность, снижает нагрузку на сервер и улучшает отзывчивость клиента.

2. Какие параметры запроса позволяют управлять пагинацией?

Основные параметры — это `page` (номер страницы) и `limit` (или `per_page`, количество элементов на странице). Например: `?page=2&limit=20`.

3. Чем полезен метод try/except?

`try/except` позволяет перехватывать и обрабатывать исключения (ошибки) в Python. При работе с API он предотвращает аварийное завершение программы из-за сетевых сбоев, ошибок HTTP, проблем с парсингом данных и т.д., делая код устойчивым к внешним сбоям.

4. Какие типы ошибок могут возникать при работе с API?

- HTTP-ошибки (4xx, 5xx коды статуса, например, 404, 500).
- Сетевые ошибки (потеря соединения, таймауты).
- Ошибки парсинга (некорректный формат данных, например, невалидный JSON).
- Логические ошибки (пустой или неполный ответ при успешном статусе 200).

5. Какие ограничения могут быть у публичных API?

- Rate limiting — ограничение количества запросов в единицу времени.

Примеры выполнения заданий

- Лимиты на объём данных (максимальное количество записей в ответе).
- Требование авторизации (API-ключ, токен).
- Ограничения по методам (разрешены только GET-запросы).
- Географические или IP-ограничения.
- Ограничения формата данных (только определённые поля или структура ответа).

Задача 1:

```
import requests

response = requests.get("https://dog.ceo/api/breeds/list/all")

all_breeds = list(response.json()["message"].keys())

limit = 5

for page in range(1, 4):
  start = (page - 1) * limit
  end = page * limit
  page_breeds = all_breeds[start:end]
  print(f"Страница {page}: {page_breeds}")
```

вывод:

Страница 1: ['affenpinscher', 'african', 'airedale', 'akita', 'appenzeller']

Страница 2: ['australian', 'bakharwal', 'basenji', 'beagle', 'bluetick']

Страница 3: ['borzoi', 'bouvier', 'boxer', 'brabancon', 'briard']

Задача 2:

```
data = {

"user": {
```

Примеры выполнения заданий

```
 "id": 123,
 "name": "Анна",
 "contacts": {
   "email": "anna@example.com",
   "phones": ["+123456789", "+987654321"]
 },
 "roles": ["admin", "editor"]
}}
email = data["user"]["contacts"]["email"]
first_phone = data["user"]["contacts"]["phones"][0]
print("Email:", email)
print("Первый телефон:", first_phone)
```

вывод:

Email: anna@example.com

Первый телефон: +123456789

Задача 3:

```
def get_city(data):
  return data.get("address", {}).get("city", "Не указано")
json_example = {
  "user": {
     "name": "Иван",
  }}
print(get_city(json_example["user"]))
```

Примеры выполнения заданий

вывод:

Не указано

Задача 4:

```
import requests

try:
  response = requests.get("https://api.publicapis.org/entries", timeout=5)
  response.raise_for_status()
  data = response.json()
  print("Успешно получили данные!")
except requests.exceptions.HTTPError as e:
  print("HTTP ошибка:", e)
except requests.exceptions.Timeout:
  print("Ошибка: запрос превысил время ожидания")
except ValueError:
  print("Ошибка: не удалось распарсить JSON")
except Exception as e:
  print("Произошла ошибка:", e)
```

вывод:

```
Произошла ошибка: HTTPSConnectionPool(host='api.publicapis.org',
port=443): Max retries exceeded with url: /entries (Caused by
NameResolutionError("<urllib3.connection.HTTPSConnection object at
0x0000017353634D60>: Failed to resolve 'api.publicapis.org' ([Errno 11001]
getaddrinfo failed)"))
```

Примеры выполнения заданий

Задача 5:

```
import requests

def extract_recursive(data, keys):
  if not keys:
    return data
  key = keys[0]
  if isinstance(data, dict) and key in data:
    return extract_recursive(data[key], keys[1:])
  return None
response = requests.get("https://dog.ceo/api/breeds/list/all")
response.raise_for_status()
json_data = response.json()
breeds = extract_recursive(json_data, ["message"])
if breeds:
  print("Породы собак:", list(breeds.keys()))
else:
  print("Данные по породам не найдены.")
```

вывод:

```
Породы собак: ['affenpinscher', 'african', 'airedale', 'akita', 'appenzeller',
'australian', 'bakharwal', 'basenji', 'beagle', 'bluetick', 'borzoi', 'bouvier', 'boxer',
'brabancon', 'briard', 'buhund', 'bulldog', 'bullterrier', 'cattledog', 'cavapoo',
'chihuahua', 'chippiparai', 'chow', 'clumber', 'cockapoo', 'collie', 'coonhound',
'corgi', 'cotondetulear', 'dachshund', 'dalmatian', 'dane', 'danish', 'deerhound',
'dhole', 'dingo', 'doberman', 'elkhound', 'entlebucher', 'eskimo', 'finnish', 'frise',
'gaddi', 'germanshepherd', 'greyhound', 'groenendael', 'havanese', 'hound',
```

Примеры выполнения заданий

'husky', 'keeshond', 'kelpie', 'kombai', 'komondor', 'kuvasz', 'labradoodle', 'labrador', 'leonberg', 'lhasa', 'malamute', 'malinois', 'maltese', 'mastiff', 'mexicanhairless', 'mix', 'mountain', 'mudhol', 'newfoundland', 'otterhound', 'ovcharka', 'papillon', 'pariah', 'pekinese', 'pembroke', 'pinscher', 'pitbull', 'pointer', 'pomeranian', 'poodle', 'pug', 'puggle', 'pyrenees', 'rajapalayam', 'redbone', 'retriever', 'ridgeback', 'rottweiler', 'saluki', 'samoyed', 'schipperke', 'schnauzer', 'segugio', 'setter', 'sharpei', 'sheepdog', 'shiba', 'shihtzu', 'spaniel', 'spitz', 'springer', 'stbernard', 'terrier', 'tervuren', 'vizsla', 'waterdog', 'weimaraner', 'whippet', 'wolfhound']

Сегодня на занятии вы:

- Познакомились с понятием пагинации.
- Научились получать данные частями из API.
- Освоили методы разбора и извлечения информации из сложных JSON-ответов.
- Узнали, как обрабатывать ошибки при работе с API и писать более устойчивый код на Python.
- Рассмотрели ограничения публичных API.

На следующем занятии вы:

- Узнаете, что такое авторизация и зачем она нужна в API.
- Познакомитесь с основными методами авторизации.
- Изучите работу с API ключами.
- Научитесь использовать токены доступа.
- Освоите передачу авторизационных данных в заголовках.
- Попрактикуетесь в работе с защищёнными API.

Получение фидбека. Рефлексия. Проверка знаний

Попросите ребят решить интерактивные задачи по теме урока

Ответы к интерактивным задачам

Практическое задание 1

Что такое пагинация в контексте работы с API?

- Способ шифрования данных при передаче.
- Метод аутентификации пользователя.
- **Способ получения данных частями, когда их слишком много для одного ответа.**
- Процесс преобразования данных из XML в JSON.

Практическое задание 2

Какая конструкция в Python является основной для безопасной обработки ошибок при выполнении запросов к API?

- if-elif-else
- for-else
- **try-except**
- while-break

Практическое задание 3

Какой формат данных чаще всего используется для обмена информацией в современных публичных API?

JSON

Ответы к интерактивным задачам

Вопрос-ответ

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