МГТУ им. Н.Э. Баумана

Домашнее Здадание

по курсу «ПиКЯП»

Тема: Создание телеграмм бота Цитатника.

Руководитель

Нардид А. Н.

29.11.2023

Студент группы ИУ5-3Б

Ильин Константин

29.11.2023

Описание Задания:

Выбор темы был сложной частью проделанной работы. Когда преподаватель дает возможность придумать тему самому, то глаза начинают разбегаться от множества идей. Ключевые слова звучали так: «программирование в удовольствие» и «что-то злободневное и полезное». Мне всегда нравится находить в жизни какие-то забавное симметрии, где кажется, что именно в этом место вселенная дала слабину, начиная от смешного анекдота, заканчивая причудливой формой снежинки. Я хочу, чтобы моя работа смогла улыбнуть своей простой идеей и забавным наполнением. Во время первого курса мы с группой решили начать собирать золотой фонд цитат преподавателей из нашего университета. Я решил что эта идея достойна второй жизни и начал написание телеграмм бота.

1)Для начала работы устанавливаем библиотеки для телеграмм бота

2)Создаем .json файл для хранения цитат

Начинаем написание

class Quote:  
 def \_\_init\_\_(self, text, author, likes=0, dislikes=0):  
 self.text = text # Текст цитаты  
 self.likes = likes # Количество лайков  
 self.dislikes = dislikes # Количество дизлайков  
  
 def to\_dict(self):  
 return {  
 "text": self.text, # Преобразует цитату в словарь  
 "likes": self.likes,  
 "dislikes": self.dislikes  
 }  
  
 def increase\_likes(self):  
 self.likes += 1 # Увеличивает количество лайков на 1  
  
 def decrease\_likes(self):  
 if self.likes > 0:  
 self.likes -= 1 # Уменьшает количество лайков на 1, если оно больше нуля  
  
 def increase\_dislikes(self):  
 self.dislikes += 1 # Увеличивает количество дизлайков на 1  
  
 def decrease\_dislikes(self):  
 if self.dislikes > 0:  
 self.dislikes -= 1 # Уменьшает количество дизлайков на 1, если оно больше нуля

`Quote` - это класс, представляющий отдельную цитату. Он содержит информацию о тексте цитаты, количестве лайков и дизлайков.

### Поля класса:

- `text`: Строка, содержащая текст цитаты.

- `likes`: Целое число, представляющее количество лайков для цитаты (по умолчанию 0).

- `dislikes`: Целое число, представляющее количество дизлайков для цитаты (по умолчанию 0).

### Методы класса:

1. `\_\_init\_\_(self, text, author, likes=0, dislikes=0)`: Конструктор класса, инициализирует объект `Quote` с заданным текстом, количеством лайков и дизлайков.

2. `to\_dict(self) -> dict`: Метод преобразует объект цитаты в словарь, содержащий текст, количество лайков и количество дизлайков.

3. `increase\_likes(self)`: Увеличивает количество лайков цитаты на 1.

4. `decrease\_likes(self)`: Уменьшает количество лайков цитаты на 1, если оно больше 0.

5. `increase\_dislikes(self)`: Увеличивает количество дизлайков цитаты на 1.

6. `decrease\_dislikes(self)`: Уменьшает количество дизлайков цитаты на 1, если оно больше 0.

### Примечания:

- `text` и `author` в параметрах конструктора не используются внутри класса, возможно, необходимо исправление или дополнение класса, чтобы учесть автора цитаты или удалить неиспользуемый параметр.

- Методы `increase\_likes`, `decrease\_likes`, `increase\_dislikes` и `decrease\_dislikes` позволяют управлять лайками и дизлайками цитаты, обеспечивая возможность увеличивать и уменьшать их количество.

class QuoteGenerator:  
  
 #Инициализация генератора цитат с категориями и текущими индексами.  
 def \_\_init\_\_(self):  
 self.categories = {}  
 self.current\_index = {}  
  
 # Загрузка существующих данных из файла JSON.  
 def load\_existing\_data(self, file\_name):  
 try:  
 with open(file\_name, 'r', encoding='utf-8') as file:  
 return json.load(file)  
 except FileNotFoundError:  
 logging.error(f"Файл '{file\_name}' не найден при чтении.")  
 except json.JSONDecodeError as e:  
 logging.error(f"Ошибка при чтении файла '{file\_name}': {e}")  
 except Exception as e:  
 logging.error(f"Неизвестная ошибка при чтении файла '{file\_name}': {e}")  
 return {}  
  
 # Сохранение данных в файл JSON.  
 def save\_data\_to\_file(self, file\_name, data):  
 try:  
 with open(file\_name, 'w', encoding='utf-8') as file:  
 json.dump(data, file, ensure\_ascii=False, indent=4)  
 except IOError as e:  
 logging.error(f"Ошибка ввода-вывода при записи в файл '{file\_name}': {e}")  
 except Exception as e:  
 logging.error(f"Неизвестная ошибка при записи в файл '{file\_name}': {e}")  
  
 #Удаление цитаты или категории из файла JSON.  
 def remove\_from\_file(self, file\_name, to\_remove, is\_category=False):  
 existing\_data = self.load\_existing\_data(file\_name)  
 try:  
 if is\_category:  
 if to\_remove in self.categories:  
 del self.categories[to\_remove]  
 else:  
 raise ValueError(f"Категория '{to\_remove}' не найдена при удалении.")  
 else:  
 found = False  
 for category, quotes in self.categories.items():  
 self.categories[category] = [quote for quote in quotes if quote.text != to\_remove]  
 if not found and any(quote.text == to\_remove for quote in quotes):  
 found = True  
 if not found:  
 raise ValueError(f"Цитата '{to\_remove}' не найдена при удалении.")  
  
 for key, quotes in self.categories.items():  
 existing\_data.setdefault(key, []).extend([quote.to\_dict() for quote in quotes])  
  
 self.save\_data\_to\_file(file\_name, existing\_data)  
  
 # После удаления категории или цитаты, перезагрузите категории  
 self.load\_quotes(file\_name)  
  
 return True  
 except ValueError as ve:  
 logging.error(f"Ошибка значения: {ve}")  
 return False  
 except Exception as e:  
 logging.error(f"Произошла ошибка при удалении: {e}")  
 return False  
  
 #Получение случайной цитаты из определенной категории.  
 def get\_random\_quote\_by\_category(self, category=None):  
 if category:  
 category = category.lower()  
 if category in self.categories and self.categories[category]:  
 if category not in self.current\_index:  
 self.current\_index[category] = 0  
  
 index = self.current\_index[category]  
 quote = self.categories[category][index]  
 self.current\_index[category] = (index + 1) % len(self.categories[category])  
 return quote  
 else:  
 return None  
 else:  
 all\_quotes = [quote for quotes in self.categories.values() for quote in quotes]  
 if all\_quotes:  
 if not self.current\_index:  
 self.current\_index = {category: 0 for category in self.categories}  
  
 category = next(iter(self.current\_index))  
 index = self.current\_index[category]  
 quote = self.categories[category][index]  
 self.current\_index[category] = (index + 1) % len(self.categories[category])  
 return quote  
 else:  
 return None  
 #Увеличение количества лайков для указанной цитаты в указанной категории.  
 def like\_quote(self, category, quote\_index):  
 if category in self.categories and quote\_index < len(self.categories[category]):  
 quote = self.categories[category][quote\_index]  
 quote.increase\_likes()  
 self.save\_quotes('quotes.json')  
 return True  
 return False  
  
  
  
 #Увеличение количества дизлайков для указанной цитаты в указанной категории.  
 def dislike\_quote(self, category, quote\_index):  
 if category in self.categories and quote\_index < len(self.categories[category]):  
 quote = self.categories[category][quote\_index]  
 quote.increase\_dislikes()  
 self.save\_quotes('quotes.json')  
 return True  
 return False  
 #Сохранение всех цитат в файл JSON.  
 def save\_quotes(self, file\_name):  
 existing\_data = {}  
 try:  
 with open(file\_name, 'r', encoding='utf-8') as file:  
 existing\_data = json.load(file)  
 except FileNotFoundError:  
 logger.error(f"Файл '{file\_name}' не найден при чтении.")  
 except json.JSONDecodeError as e:  
 logger.error(f"Ошибка при чтении файла '{file\_name}': {e}")  
 except Exception as e:  
 logger.error(f"Неизвестная ошибка при чтении файла '{file\_name}': {e}")  
  
 updated\_data = {} # Создаем новый словарь для обновленных данных  
  
 # Обновление данных о лайках и дислайках перед сохранением  
 for key, quotes in self.categories.items():  
 updated\_quotes = []  
 for quote in quotes:  
 existing\_quote = next((q for q in existing\_data.get(key, []) if q['text'] == quote.text), None)  
 if existing\_quote:  
 # Найденная цитата в существующих данных - обновим ее лайки и дислайки  
 existing\_quote['likes'] = quote.likes  
 existing\_quote['dislikes'] = quote.dislikes  
 updated\_quotes.append(existing\_quote)  
 else:  
 # Цитата не найдена - добавим ее в обновленные данные  
 updated\_quotes.append(quote.to\_dict())  
 updated\_data[key] = updated\_quotes  
  
 try:  
 with open(file\_name, 'w', encoding='utf-8') as file:  
 json.dump(updated\_data, file, ensure\_ascii=False, indent=4)  
 except IOError as e:  
 logger.error(f"Ошибка ввода-вывода при записи в файл '{file\_name}': {e}")  
 except Exception as e:  
 logger.error(f"Неизвестная ошибка при записи в файл '{file\_name}': {e}")  
  
 #Сохранение всех цитат в файл JSON.  
 def add\_quote(self, quote\_type, new\_quote):  
 if quote\_type and new\_quote:  
 quote\_type = quote\_type.strip().lower()  
 new\_quote = new\_quote.strip()  
  
 # Проверяем наличие цитаты в категории  
 if any(quote.text == new\_quote for quote in self.categories.get(quote\_type, [])):  
 logger.info("Цитата уже существует в данной категории.")  
 return # Цитата уже существует, не добавляем её повторно  
  
 # Добавление цитаты в категорию  
 if quote\_type in self.categories:  
 self.categories[quote\_type].append(Quote(new\_quote, quote\_type))  
 else:  
 logger.info(f"Цитата не попадает в существующую категорию.")  
 response = input(f"Цитата '{new\_quote}' не попадает в существующую категорию '{quote\_type}'. "  
 "Что с ней делать? (Создать новую категорию / Проигнорировать / Другое) ")  
 # Здесь вы можете обработать ответ пользователя  
 # Например, создать новую категорию или проигнорировать цитату  
 if response.lower() == "создать новую категорию":  
 self.categories[quote\_type] = [Quote(new\_quote, quote\_type)]  
 elif response.lower() == "проигнорировать":  
 pass # Ничего не делаем с цитатой  
 else:  
 # Другие действия с цитатой, например, добавление в "unknown" или другую обработку  
 pass  
  
 # Сохранение цитат после добавления  
 self.save\_quotes('quotes.json')  
 else:  
 logger.error("Используйте формат /addquote НАЗВАНИЕ\_КАТЕГОРИИ: ТЕКСТ\_ЦИТАТЫ")  
  
 #Получение цитаты из указанной категории.  
 def get\_quotes\_by\_category(self, category):  
 category = category.lower()  
 if category in self.categories and self.categories[category]:  
 if category not in self.current\_index:  
 self.current\_index[category] = 0  
  
 index = self.current\_index[category]  
 if index < len(self.categories[category]):  
 quote = self.categories[category][index].text  
 self.current\_index[category] = (index + 1) % len(self.categories[category])  
 return quote  
 else:  
 error\_message = "Достигнут конец списка цитат для данной категории."  
 logging.error(error\_message)  
 return error\_message  
 else:  
 error\_message = f"Цитаты для категории '{category}' не найдены."  
 logging.error(error\_message)  
 return error\_message  
  
 #Загрузка цитат из файла JSON.  
 def load\_quotes(self, file\_name):  
 try:  
 with open(file\_name, 'r', encoding='utf-8') as file:  
 quotes\_data = json.load(file)  
 except FileNotFoundError:  
 quotes\_data = {}  
 logging.error(f"Файл '{file\_name}' не найден при чтении.")  
 except json.JSONDecodeError as e:  
 quotes\_data = {}  
 logging.error(f"Ошибка при чтении файла '{file\_name}': {e}")  
 except Exception as e:  
 quotes\_data = {}  
 logging.error(f"Неизвестная ошибка при чтении файла '{file\_name}': {e}")  
 else:  
 for category, quotes in quotes\_data.items():  
 self.categories[category] = []  
 for quote in quotes:  
 # Добавьте проверку наличия ключа 'author' в цитате  
 author = quote.get('author', None)  
 # Используйте get() для безопасного получения значения по ключу  
 # Если 'author' отсутствует, установите значение по умолчанию (None или что-то другое)  
 self.categories[category].append(Quote(  
 quote['text'], author, quote['likes'], quote['dislikes']  
 ))  
 # удаление поторяющихся цитат  
 def remove\_duplicate\_quotes(self, file\_name):  
 # Загрузка цитат из файла  
 self.load\_quotes(file\_name)  
  
 # Проход по каждой категории  
 for category, quotes in self.categories.items():  
 existing\_quotes = set()  
 unique\_quotes = []  
  
 # Проверка на уникальность цитат в категории  
 for quote in quotes:  
 if quote.text not in existing\_quotes:  
 existing\_quotes.add(quote.text)  
 unique\_quotes.append(quote)  
 else:  
 # Логирование повторяющейся цитаты  
 print(f"Найдена повторяющаяся цитата в категории '{category}': '{quote.text}'")  
  
 # Замена цитат в категории уникальными цитатами  
 self.categories[category] = unique\_quotes  
  
 # Сохранение обновленных данных в файл  
 self.save\_quotes(file\_name)  
 print("Проверка сохранения изменений после удаления повторяющихся цитат:")  
 print(self.categories) # Вывод данных для проверки   
  
 #Получение случайной цитаты из категории, указанной в сообщении пользователя.  
 def get\_quote\_by\_category(update, context):  
 from random import choice  
  
 category = context.args[0].lower() if context.args else None  
 quotes\_in\_category = quote\_generator.categories.get(category)  
  
 if quotes\_in\_category:  
 random\_quote = choice(quotes\_in\_category).text  
 context.chat\_data["last\_quote"] = random\_quote  
 update.message.reply\_text(f"Случайная цитата из категории '{category}': {random\_quote}")  
 update.message.reply\_text("Поставьте лайк этой цитате, используя команду /like.")  
 update.message.reply\_text("Поставьте дизлайк этой цитате, используя команду /dislike.")  
 else:  
 update.message.reply\_text(f"Отсутствуют цитаты для категории '{category}'.")  
  
 #Сброс количества лайков и дизлайков для всех цитат.  
 def reset\_likes\_and\_dislikes(self):  
 for category in self.categories.values():  
 for quote in category:  
 quote.likes = 0  
 quote.dislikes = 0  
 self.save\_quotes('quotes.json')  
  
  
 #Получение случайной цитаты из всех доступных цитат.  
 def get\_random\_quote(self):  
 all\_quotes = [quote for quotes in self.categories.values() for quote in quotes]  
 if all\_quotes:  
 random\_quote = random.choice(all\_quotes)  
 return random\_quote.text  
 else:  
 return "Нет доступных цитат."  
  
quote\_generator = QuoteGenerator() #создает экземпляр класса  
quote\_generator.reset\_likes\_and\_dislikes() #выполняет сброс количества лайков и дизлайков для всех цитат, хранящихся в системе.  
quote\_generator.remove\_duplicate\_quotes('quotes.json')

`QuoteGenerator` - это класс, предназначенный для управления цитатами, их хранения, добавления, удаления, а также получения случайных цитат из различных категорий.

### Поля класса:

- `categories`: Словарь, содержащий категории цитат, каждая из которых представлена списком объектов цитат.

- `current\_index`: Словарь, хранящий текущие индексы для каждой категории цитат.

### Основные методы класса:

1. `load\_existing\_data(self, file\_name)`: Загружает существующие данные из файла JSON.

2. `save\_data\_to\_file(self, file\_name, data)`: Сохраняет данные в файл JSON.

3. `remove\_from\_file(self, file\_name, to\_remove, is\_category=False)`: Удаляет цитату или категорию из файла JSON.

4. `get\_random\_quote\_by\_category(self, category=None)`: Получает случайную цитату из определенной категории.

5. `like\_quote(self, category, quote\_index)`: Увеличивает количество лайков для указанной цитаты в указанной категории.

6. `dislike\_quote(self, category, quote\_index)`: Увеличивает количество дизлайков для указанной цитаты в указанной категории.

7. `save\_quotes(self, file\_name)`: Сохраняет все цитаты в файл JSON.

8. `add\_quote(self, quote\_type, new\_quote)`: Добавляет цитату в указанную категорию.

9. `load\_quotes(self, file\_name)`: Загружает цитаты из файла JSON.

10. `remove\_duplicate\_quotes(self, file\_name)`: Удаляет повторяющиеся цитаты из системы.

11. `reset\_likes\_and\_dislikes(self)`: Сбрасывает количество лайков и дизлайков для всех цитат в системе.

12. `get\_random\_quote(self)`: Получает случайную цитату из всех доступных цитат.

# Функции добавления и получения цитат:  
def add\_quote(update: Update, context: CallbackContext) -> None:  
 try:  
 user\_input = None  
 if update.message and update.message.text:  
 user\_input = update.message.text.split(' ', 1)  
  
 if user\_input is not None and len(user\_input) == 2:  
 if ':' in user\_input[1]:  
 quote\_type, new\_quote = user\_input[1].split(':', 1)  
 quote\_type = quote\_type.strip().lower()  
 new\_quote = new\_quote.strip()  
  
 if not quote\_type or not new\_quote:  
 raise ValueError("Необходимо указать и категорию, и цитату.")  
  
 if quote\_type in quote\_generator.categories:  
 quote\_generator.add\_quote(quote\_type, new\_quote)  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Цитата успешно добавлена в категорию '{quote\_type}'.")  
 else:  
 quote\_generator.categories[quote\_type] = [Quote(new\_quote, 'unknown')]  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Категория '{quote\_type}' создана, цитата успешно добавлена.")  
 else:  
 update.message.reply\_text("Используйте формат /addquote НАЗВАНИЕ\_КАТЕГОРИИ: ТЕКСТ\_ЦИТАТЫ")  
 else:  
 update.message.reply\_text("Используйте формат /addquote НАЗВАНИЕ\_КАТЕГОРИИ: ТЕКСТ\_ЦИТАТЫ")  
  
 except ValueError as ve:  
 logger.error(f"Ошибка значения: {ve}")  
 update.message.reply\_text(str(ve))  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при добавлении цитаты. Пожалуйста, попробуйте еще раз.")  
def get\_quote(update: Update, context: CallbackContext):  
 category = context.args[0].lower() if context.args else None  
 category\_text = f"из категории '{category}'" if category else "без категории"  
  
 random\_quote = quote\_generator.get\_random\_quote\_by\_category(category)  
 if random\_quote:  
 context.user\_data["last\_quote"] = random\_quote.text  
 update.message.reply\_text(f"Цитата {category\_text}: {random\_quote.text}\n\n"  
 "Поставьте лайк этой цитате, используя команду /like.\n"  
 "Поставьте дизлайк этой цитате, используя команду /dislike.")  
 else:  
 update.message.reply\_text("Отсутствуют цитаты для вывода.")  
  
# Функции работы с категориями и списком категорий:  
def category\_list(update, context):  
 categories = [f"{category}" for category in quote\_generator.categories]  
 categories\_text = "\n".join(categories)  
 update.message.reply\_text(f"Доступные категории:\n{categories\_text}")  
def delete\_category\_command(update: Update, context: CallbackContext):  
 if not context.args:  
 update.message.reply\_text("Укажите категорию для удаления.")  
 return  
  
 to\_remove = context.args[0].lower()  
 if to\_remove in quote\_generator.categories:  
 del quote\_generator.categories[to\_remove]  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Категория '{to\_remove}' успешно удалена.")  
 else:  
 update.message.reply\_text(f"Категория '{to\_remove}' не найдена.")  
  
  
# Функции сохранения данных:  
  
def save\_quotes\_to\_file(file\_name, categories):  
  
 serialized\_categories = {  
 key: [quote.to\_dict() for quote in quotes] for key, quotes in categories.items()  
 }  
  
  
 with open(file\_name, 'w', encoding='utf-8') as file:  
 json.dump(serialized\_categories, file, ensure\_ascii=False, indent=4)  
  
  
  
# Функции работы с лайками и дизлайками:  
def like(update: Update, context: CallbackContext):  
 update\_quote\_status(update, context, "лайк")  
  
def dislike(update: Update, context: CallbackContext):  
 update\_quote\_status(update, context, "дизлайк")  
  
def update\_quote\_status(update: Update, context: CallbackContext, action: str):  
 try:  
 text = update.message.text  
 last\_quote = context.user\_data.get("last\_quote")  
  
 print(f"Полученный текст: {text}")  
 print(f"Последняя цитата: {last\_quote}")  
  
 if text not in ["/like", "/dislike"]:  
 update.message.reply\_text("Пожалуйста, используйте /like или /dislike для оценки цитаты.")  
 return  
  
 if not last\_quote:  
 update.message.reply\_text("Кажется, вы еще не получили цитату. Используйте /getquote [категория] сначала.")  
 return  
  
 found\_quote = None  
 for category\_quotes in quote\_generator.categories.values():  
 for quote in category\_quotes:  
 if quote.text.lower() == last\_quote.lower():  
 found\_quote = quote  
 break  
 if found\_quote:  
 break  
  
 if not found\_quote:  
 update.message.reply\_text("Цитата не найдена.")  
 return  
  
 if text == "/like":  
 found\_quote.increase\_likes()  
 print(f"Лайк для цитаты '{last\_quote}' увеличен.")  
 else:  
 found\_quote.increase\_dislikes()  
 print(f"Дизлайк для цитаты '{last\_quote}' увеличен.")  
  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Цитата '{last\_quote}' {action}нута.")  
  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при обновлении статуса цитаты.")  
  
# Функции удаления цитат:  
  
def delete\_quote(update: Update, context: CallbackContext):  
 if not context.args:  
 update.message.reply\_text("Укажите текст цитаты для удаления.")  
 return  
  
 to\_remove = ' '.join(context.args)  
 remove\_quote\_or\_category(update, context, to\_remove)  
  
def confirm\_deletion(update, context, to\_remove):  
 keyboard = [['Да', 'Нет']]  
 context.user\_data['delete\_quote'] = to\_remove # Сохраняем цитату для удаления в данных пользователя  
 update.message.reply\_text(  
 "Вы уверены, что хотите удалить цитату?",  
 reply\_markup=ReplyKeyboardMarkup(keyboard, one\_time\_keyboard=True),  
 )  
  
def remove\_quote\_or\_category(update: Update, context: CallbackContext, to\_remove: str, is\_category: bool = False):  
 if not to\_remove:  
 update.message.reply\_text(f"Укажите {'категорию' if is\_category else 'текст цитаты'} для удаления.")  
 return  
  
 removed = False # Флаг для отслеживания удаления  
  
 # Удаляем категорию  
 if is\_category:  
 if to\_remove.lower() in quote\_generator.categories:  
 del quote\_generator.categories[to\_remove.lower()]  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Категория '{to\_remove}' успешно удалена.")  
 else:  
 update.message.reply\_text(f"Категория '{to\_remove}' не найдена.")  
 else:  
 # Удаляем цитату из категорий  
 removed = False  
 for category, quotes in quote\_generator.categories.items():  
 updated\_quotes = quotes.copy() # Создаем копию списка цитат  
  
 updated\_quotes = [quote for quote in updated\_quotes if quote.text.lower() != to\_remove.lower()]  
  
 if len(updated\_quotes) < len(quotes):  
 removed = True  
 quote\_generator.categories[category] = updated\_quotes  
 quote\_texts = [quote.text for quote in updated\_quotes]  
 print(f"Категория: {category}, Цитаты: {quote\_texts}")  
  
 if not removed:  
 update.message.reply\_text(f"Цитата '{to\_remove}' не найдена в категориях.")  
 else:  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Цитата '{to\_remove}' успешно удалена из категорий.")  
  
 # Применение функции  
 remove\_quote\_or\_category(update, context, 'текст\_цитаты\_для\_удаления')  
  
  
# Функции анализа цитат:  
def analyze\_quotes(update, context):  
 try:  
 all\_quotes = [quote for quotes in quote\_generator.categories.values() for quote in quotes]  
 sorted\_quotes = sorted(all\_quotes, key=lambda x: x.likes / max(1, x.dislikes), reverse=True)  
  
 top\_5\_quotes = sorted\_quotes[:5]  
 message = "Топ 5 цитат:\n"  
 for index, quote in enumerate(top\_5\_quotes, start=1):  
 message += f"{index}. '{quote.text}' - Лайков: {quote.likes}, Дизлайков: {quote.dislikes}\n"  
  
 update.message.reply\_text(message)  
  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при анализе цитат.")  
  
  
  
# Функции управления командами:  
def start(update: Update, context: CallbackContext) -> None:  
 existing\_commands = ["/start", "/addquote", "/deletequote", "/deletecategory", "/category", "/resetlikes", "/analyze"]  
 all\_commands\_text = '\n'.join(existing\_commands)  
 update.message.reply\_text(f"Привет! Чтобы получить цитату, используйте команды:\n{all\_commands\_text}")  
def sanitize\_command\_name(command):  
 return ''.join(c.lower() for c in command if c.isalnum())  
  
def is\_valid\_command(command):  
 return bool(re.match(r'^[A-Za-z0-9\_]+$', command))  
  
from telegram import Update, ReplyKeyboardMarkup  
from telegram.ext import CallbackContext  
def reset\_likes\_and\_dislikes(update: Update, context: CallbackContext):  
 try:  
 if not context.bot.get\_chat\_member(update.message.chat\_id, update.message.from\_user.id).can\_promote\_members:  
 update.message.reply\_text("У вас нет прав для выполнения этой операции.")  
 return  
  
 quote\_generator.reset\_likes\_and\_dislikes()  
 quote\_generator.save\_quotes('quotes.json')  
  
 update.message.reply\_text("Лайки и дизлайки для всех цитат были обнулены.")  
  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при сбросе лайков и дизлайков.")

### Функции добавления и получения цитат:

1. `add\_quote(update, context)`: Добавляет цитату в указанную категорию или создает новую категорию.

2. `get\_quote(update, context)`: Получает случайную цитату из указанной категории или из всех доступных.

### Функции работы с категориями и списком категорий:

1. `category\_list(update, context)`: Отображает список доступных категорий.

2. `delete\_category\_command(update, context)`: Удаляет указанную категорию.

### Функции сохранения данных:

1. `save\_quotes\_to\_file(file\_name, categories)`: Сохраняет цитаты в файл JSON.

### Функции работы с лайками и дизлайками:

1. `like(update, context)`: Увеличивает количество лайков для последней полученной цитаты.

2. `dislike(update, context)`: Увеличивает количество дизлайков для последней полученной цитаты.

### Функции удаления цитат:

1. `delete\_quote(update, context)`: Удаляет указанную цитату.

2. `confirm\_deletion(update, context, to\_remove)`: Запрашивает подтверждение на удаление цитаты.

3. `remove\_quote\_or\_category(update, context, to\_remove)`: Удаляет цитату или категорию.

### Функции анализа цитат:

1. `analyze\_quotes(update, context)`: Анализирует цитаты и выводит топ-5 цитат по популярности.

### Функции управления командами:

1. `start(update, context)`: Выводит список доступных команд для начала использования бота.

2. `sanitize\_command\_name(command)`: Очищает команду от недопустимых символов.

3. `is\_valid\_command(command)`: Проверяет допустимость команды.

### Дополнительные функции:

1. `reset\_likes\_and\_dislikes(update, context)`: Сбрасывает количество лайков и дизлайков для всех цитат.

Весь код:

import random  
import json  
import re  
from telegram import Update  
from telegram.ext import CallbackContext, Updater, CommandHandler, MessageHandler, Filters  
from telegram import ReplyKeyboardMarkup  
import logging  
  
# Настройка логгирования  
logging.basicConfig(  
 level=logging.INFO,  
 format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'  
)  
logger = logging.getLogger(\_\_name\_\_)  
  
class Quote:  
 def \_\_init\_\_(self, text, author, likes=0, dislikes=0):  
 self.text = text # Текст цитаты  
 self.likes = likes # Количество лайков  
 self.dislikes = dislikes # Количество дизлайков  
  
 def to\_dict(self):  
 return {  
 "text": self.text, # Преобразует цитату в словарь  
 "likes": self.likes,  
 "dislikes": self.dislikes  
 }  
  
 def increase\_likes(self):  
 self.likes += 1 # Увеличивает количество лайков на 1  
  
 def decrease\_likes(self):  
 if self.likes > 0:  
 self.likes -= 1 # Уменьшает количество лайков на 1, если оно больше нуля  
  
 def increase\_dislikes(self):  
 self.dislikes += 1 # Увеличивает количество дизлайков на 1  
  
 def decrease\_dislikes(self):  
 if self.dislikes > 0:  
 self.dislikes -= 1 # Уменьшает количество дизлайков на 1, если оно больше нуля  
  
  
  
  
class QuoteGenerator:  
  
 #Инициализация генератора цитат с категориями и текущими индексами.  
 def \_\_init\_\_(self):  
 self.categories = {}  
 self.current\_index = {}  
  
 # Загрузка существующих данных из файла JSON.  
 def load\_existing\_data(self, file\_name):  
 try:  
 with open(file\_name, 'r', encoding='utf-8') as file:  
 return json.load(file)  
 except FileNotFoundError:  
 logging.error(f"Файл '{file\_name}' не найден при чтении.")  
 except json.JSONDecodeError as e:  
 logging.error(f"Ошибка при чтении файла '{file\_name}': {e}")  
 except Exception as e:  
 logging.error(f"Неизвестная ошибка при чтении файла '{file\_name}': {e}")  
 return {}  
  
 # Сохранение данных в файл JSON.  
 def save\_data\_to\_file(self, file\_name, data):  
 try:  
 with open(file\_name, 'w', encoding='utf-8') as file:  
 json.dump(data, file, ensure\_ascii=False, indent=4)  
 except IOError as e:  
 logging.error(f"Ошибка ввода-вывода при записи в файл '{file\_name}': {e}")  
 except Exception as e:  
 logging.error(f"Неизвестная ошибка при записи в файл '{file\_name}': {e}")  
  
 #Удаление цитаты или категории из файла JSON.  
 def remove\_from\_file(self, file\_name, to\_remove, is\_category=False):  
 existing\_data = self.load\_existing\_data(file\_name)  
 try:  
 if is\_category:  
 if to\_remove in self.categories:  
 del self.categories[to\_remove]  
 else:  
 raise ValueError(f"Категория '{to\_remove}' не найдена при удалении.")  
 else:  
 found = False  
 for category, quotes in self.categories.items():  
 self.categories[category] = [quote for quote in quotes if quote.text != to\_remove]  
 if not found and any(quote.text == to\_remove for quote in quotes):  
 found = True  
 if not found:  
 raise ValueError(f"Цитата '{to\_remove}' не найдена при удалении.")  
  
 for key, quotes in self.categories.items():  
 existing\_data.setdefault(key, []).extend([quote.to\_dict() for quote in quotes])  
  
 self.save\_data\_to\_file(file\_name, existing\_data)  
  
 # После удаления категории или цитаты, перезагрузите категории  
 self.load\_quotes(file\_name)  
  
 return True  
 except ValueError as ve:  
 logging.error(f"Ошибка значения: {ve}")  
 return False  
 except Exception as e:  
 logging.error(f"Произошла ошибка при удалении: {e}")  
 return False  
  
 #Получение случайной цитаты из определенной категории.  
 def get\_random\_quote\_by\_category(self, category=None):  
 if category:  
 category = category.lower()  
 if category in self.categories and self.categories[category]:  
 if category not in self.current\_index:  
 self.current\_index[category] = 0  
  
 index = self.current\_index[category]  
 quote = self.categories[category][index]  
 self.current\_index[category] = (index + 1) % len(self.categories[category])  
 return quote  
 else:  
 return None  
 else:  
 all\_quotes = [quote for quotes in self.categories.values() for quote in quotes]  
 if all\_quotes:  
 if not self.current\_index:  
 self.current\_index = {category: 0 for category in self.categories}  
  
 category = next(iter(self.current\_index))  
 index = self.current\_index[category]  
 quote = self.categories[category][index]  
 self.current\_index[category] = (index + 1) % len(self.categories[category])  
 return quote  
 else:  
 return None  
 #Увеличение количества лайков для указанной цитаты в указанной категории.  
 def like\_quote(self, category, quote\_index):  
 if category in self.categories and quote\_index < len(self.categories[category]):  
 quote = self.categories[category][quote\_index]  
 quote.increase\_likes()  
 self.save\_quotes('quotes.json')  
 return True  
 return False  
  
  
  
 #Увеличение количества дизлайков для указанной цитаты в указанной категории.  
 def dislike\_quote(self, category, quote\_index):  
 if category in self.categories and quote\_index < len(self.categories[category]):  
 quote = self.categories[category][quote\_index]  
 quote.increase\_dislikes()  
 self.save\_quotes('quotes.json')  
 return True  
 return False  
 #Сохранение всех цитат в файл JSON.  
 def save\_quotes(self, file\_name):  
 existing\_data = {}  
 try:  
 with open(file\_name, 'r', encoding='utf-8') as file:  
 existing\_data = json.load(file)  
 except FileNotFoundError:  
 logger.error(f"Файл '{file\_name}' не найден при чтении.")  
 except json.JSONDecodeError as e:  
 logger.error(f"Ошибка при чтении файла '{file\_name}': {e}")  
 except Exception as e:  
 logger.error(f"Неизвестная ошибка при чтении файла '{file\_name}': {e}")  
  
 updated\_data = {} # Создаем новый словарь для обновленных данных  
  
 # Обновление данных о лайках и дислайках перед сохранением  
 for key, quotes in self.categories.items():  
 updated\_quotes = []  
 for quote in quotes:  
 existing\_quote = next((q for q in existing\_data.get(key, []) if q['text'] == quote.text), None)  
 if existing\_quote:  
 # Найденная цитата в существующих данных - обновим ее лайки и дислайки  
 existing\_quote['likes'] = quote.likes  
 existing\_quote['dislikes'] = quote.dislikes  
 updated\_quotes.append(existing\_quote)  
 else:  
 # Цитата не найдена - добавим ее в обновленные данные  
 updated\_quotes.append(quote.to\_dict())  
 updated\_data[key] = updated\_quotes  
  
 try:  
 with open(file\_name, 'w', encoding='utf-8') as file:  
 json.dump(updated\_data, file, ensure\_ascii=False, indent=4)  
 except IOError as e:  
 logger.error(f"Ошибка ввода-вывода при записи в файл '{file\_name}': {e}")  
 except Exception as e:  
 logger.error(f"Неизвестная ошибка при записи в файл '{file\_name}': {e}")  
  
 #Сохранение всех цитат в файл JSON.  
 def add\_quote(self, quote\_type, new\_quote):  
 if quote\_type and new\_quote:  
 quote\_type = quote\_type.strip().lower()  
 new\_quote = new\_quote.strip()  
  
 # Проверяем наличие цитаты в категории  
 if any(quote.text == new\_quote for quote in self.categories.get(quote\_type, [])):  
 logger.info("Цитата уже существует в данной категории.")  
 return # Цитата уже существует, не добавляем её повторно  
  
 # Добавление цитаты в категорию  
 if quote\_type in self.categories:  
 self.categories[quote\_type].append(Quote(new\_quote, quote\_type))  
 else:  
 logger.info(f"Цитата не попадает в существующую категорию.")  
 response = input(f"Цитата '{new\_quote}' не попадает в существующую категорию '{quote\_type}'. "  
 "Что с ней делать? (Создать новую категорию / Проигнорировать / Другое) ")  
 # Здесь вы можете обработать ответ пользователя  
 # Например, создать новую категорию или проигнорировать цитату  
 if response.lower() == "создать новую категорию":  
 self.categories[quote\_type] = [Quote(new\_quote, quote\_type)]  
 elif response.lower() == "проигнорировать":  
 pass # Ничего не делаем с цитатой  
 else:  
 # Другие действия с цитатой, например, добавление в "unknown" или другую обработку  
 pass  
  
 # Сохранение цитат после добавления  
 self.save\_quotes('quotes.json')  
 else:  
 logger.error("Используйте формат /addquote НАЗВАНИЕ\_КАТЕГОРИИ: ТЕКСТ\_ЦИТАТЫ")  
  
 #Получение цитаты из указанной категории.  
 def get\_quotes\_by\_category(self, category):  
 category = category.lower()  
 if category in self.categories and self.categories[category]:  
 if category not in self.current\_index:  
 self.current\_index[category] = 0  
  
 index = self.current\_index[category]  
 if index < len(self.categories[category]):  
 quote = self.categories[category][index].text  
 self.current\_index[category] = (index + 1) % len(self.categories[category])  
 return quote  
 else:  
 error\_message = "Достигнут конец списка цитат для данной категории."  
 logging.error(error\_message)  
 return error\_message  
 else:  
 error\_message = f"Цитаты для категории '{category}' не найдены."  
 logging.error(error\_message)  
 return error\_message  
  
 #Загрузка цитат из файла JSON.  
 def load\_quotes(self, file\_name):  
 try:  
 with open(file\_name, 'r', encoding='utf-8') as file:  
 quotes\_data = json.load(file)  
 except FileNotFoundError:  
 quotes\_data = {}  
 logging.error(f"Файл '{file\_name}' не найден при чтении.")  
 except json.JSONDecodeError as e:  
 quotes\_data = {}  
 logging.error(f"Ошибка при чтении файла '{file\_name}': {e}")  
 except Exception as e:  
 quotes\_data = {}  
 logging.error(f"Неизвестная ошибка при чтении файла '{file\_name}': {e}")  
 else:  
 for category, quotes in quotes\_data.items():  
 self.categories[category] = []  
 for quote in quotes:  
 # Добавьте проверку наличия ключа 'author' в цитате  
 author = quote.get('author', None)  
 # Используйте get() для безопасного получения значения по ключу  
 # Если 'author' отсутствует, установите значение по умолчанию (None или что-то другое)  
 self.categories[category].append(Quote(  
 quote['text'], author, quote['likes'], quote['dislikes']  
 ))  
 # удаление поторяющихся цитат  
 def remove\_duplicate\_quotes(self, file\_name):  
 # Загрузка цитат из файла  
 self.load\_quotes(file\_name)  
  
 # Проход по каждой категории  
 for category, quotes in self.categories.items():  
 existing\_quotes = set()  
 unique\_quotes = []  
  
 # Проверка на уникальность цитат в категории  
 for quote in quotes:  
 if quote.text not in existing\_quotes:  
 existing\_quotes.add(quote.text)  
 unique\_quotes.append(quote)  
 else:  
 # Логирование повторяющейся цитаты  
 print(f"Найдена повторяющаяся цитата в категории '{category}': '{quote.text}'")  
  
 # Замена цитат в категории уникальными цитатами  
 self.categories[category] = unique\_quotes  
  
 # Сохранение обновленных данных в файл  
 self.save\_quotes(file\_name)  
 print("Проверка сохранения изменений после удаления повторяющихся цитат:")  
 print(self.categories) # Вывод данных для проверки   
  
 #Получение случайной цитаты из категории, указанной в сообщении пользователя.  
 def get\_quote\_by\_category(update, context):  
 from random import choice  
  
 category = context.args[0].lower() if context.args else None  
 quotes\_in\_category = quote\_generator.categories.get(category)  
  
 if quotes\_in\_category:  
 random\_quote = choice(quotes\_in\_category).text  
 context.chat\_data["last\_quote"] = random\_quote  
 update.message.reply\_text(f"Случайная цитата из категории '{category}': {random\_quote}")  
 update.message.reply\_text("Поставьте лайк этой цитате, используя команду /like.")  
 update.message.reply\_text("Поставьте дизлайк этой цитате, используя команду /dislike.")  
 else:  
 update.message.reply\_text(f"Отсутствуют цитаты для категории '{category}'.")  
  
 #Сброс количества лайков и дизлайков для всех цитат.  
 def reset\_likes\_and\_dislikes(self):  
 for category in self.categories.values():  
 for quote in category:  
 quote.likes = 0  
 quote.dislikes = 0  
 self.save\_quotes('quotes.json')  
  
  
 #Получение случайной цитаты из всех доступных цитат.  
 def get\_random\_quote(self):  
 all\_quotes = [quote for quotes in self.categories.values() for quote in quotes]  
 if all\_quotes:  
 random\_quote = random.choice(all\_quotes)  
 return random\_quote.text  
 else:  
 return "Нет доступных цитат."  
  
quote\_generator = QuoteGenerator() #создает экземпляр класса  
quote\_generator.reset\_likes\_and\_dislikes() #выполняет сброс количества лайков и дизлайков для всех цитат, хранящихся в системе.  
quote\_generator.remove\_duplicate\_quotes('quotes.json')  
  
  
# Функции добавления и получения цитат:  
def add\_quote(update: Update, context: CallbackContext) -> None:  
 try:  
 user\_input = None  
 if update.message and update.message.text:  
 user\_input = update.message.text.split(' ', 1)  
  
 if user\_input is not None and len(user\_input) == 2:  
 if ':' in user\_input[1]:  
 quote\_type, new\_quote = user\_input[1].split(':', 1)  
 quote\_type = quote\_type.strip().lower()  
 new\_quote = new\_quote.strip()  
  
 if not quote\_type or not new\_quote:  
 raise ValueError("Необходимо указать и категорию, и цитату.")  
  
 if quote\_type in quote\_generator.categories:  
 quote\_generator.add\_quote(quote\_type, new\_quote)  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Цитата успешно добавлена в категорию '{quote\_type}'.")  
 else:  
 quote\_generator.categories[quote\_type] = [Quote(new\_quote, 'unknown')]  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Категория '{quote\_type}' создана, цитата успешно добавлена.")  
 else:  
 update.message.reply\_text("Используйте формат /addquote НАЗВАНИЕ\_КАТЕГОРИИ: ТЕКСТ\_ЦИТАТЫ")  
 else:  
 update.message.reply\_text("Используйте формат /addquote НАЗВАНИЕ\_КАТЕГОРИИ: ТЕКСТ\_ЦИТАТЫ")  
  
 except ValueError as ve:  
 logger.error(f"Ошибка значения: {ve}")  
 update.message.reply\_text(str(ve))  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при добавлении цитаты. Пожалуйста, попробуйте еще раз.")  
def get\_quote(update: Update, context: CallbackContext):  
 category = context.args[0].lower() if context.args else None  
 category\_text = f"из категории '{category}'" if category else "без категории"  
  
 random\_quote = quote\_generator.get\_random\_quote\_by\_category(category)  
 if random\_quote:  
 context.user\_data["last\_quote"] = random\_quote.text  
 update.message.reply\_text(f"Цитата {category\_text}: {random\_quote.text}\n\n"  
 "Поставьте лайк этой цитате, используя команду /like.\n"  
 "Поставьте дизлайк этой цитате, используя команду /dislike.")  
 else:  
 update.message.reply\_text("Отсутствуют цитаты для вывода.")  
  
# Функции работы с категориями и списком категорий:  
def category\_list(update, context):  
 categories = [f"{category}" for category in quote\_generator.categories]  
 categories\_text = "\n".join(categories)  
 update.message.reply\_text(f"Доступные категории:\n{categories\_text}")  
def delete\_category\_command(update: Update, context: CallbackContext):  
 if not context.args:  
 update.message.reply\_text("Укажите категорию для удаления.")  
 return  
  
 to\_remove = context.args[0].lower()  
 if to\_remove in quote\_generator.categories:  
 del quote\_generator.categories[to\_remove]  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Категория '{to\_remove}' успешно удалена.")  
 else:  
 update.message.reply\_text(f"Категория '{to\_remove}' не найдена.")  
  
  
# Функции сохранения данных:  
  
def save\_quotes\_to\_file(file\_name, categories):  
  
 serialized\_categories = {  
 key: [quote.to\_dict() for quote in quotes] for key, quotes in categories.items()  
 }  
  
  
 with open(file\_name, 'w', encoding='utf-8') as file:  
 json.dump(serialized\_categories, file, ensure\_ascii=False, indent=4)  
  
  
  
# Функции работы с лайками и дизлайками:  
def like(update: Update, context: CallbackContext):  
 update\_quote\_status(update, context, "лайк")  
  
def dislike(update: Update, context: CallbackContext):  
 update\_quote\_status(update, context, "дизлайк")  
  
def update\_quote\_status(update: Update, context: CallbackContext, action: str):  
 try:  
 text = update.message.text  
 last\_quote = context.user\_data.get("last\_quote")  
  
 print(f"Полученный текст: {text}")  
 print(f"Последняя цитата: {last\_quote}")  
  
 if text not in ["/like", "/dislike"]:  
 update.message.reply\_text("Пожалуйста, используйте /like или /dislike для оценки цитаты.")  
 return  
  
 if not last\_quote:  
 update.message.reply\_text("Кажется, вы еще не получили цитату. Используйте /getquote [категория] сначала.")  
 return  
  
 found\_quote = None  
 for category\_quotes in quote\_generator.categories.values():  
 for quote in category\_quotes:  
 if quote.text.lower() == last\_quote.lower():  
 found\_quote = quote  
 break  
 if found\_quote:  
 break  
  
 if not found\_quote:  
 update.message.reply\_text("Цитата не найдена.")  
 return  
  
 if text == "/like":  
 found\_quote.increase\_likes()  
 print(f"Лайк для цитаты '{last\_quote}' увеличен.")  
 else:  
 found\_quote.increase\_dislikes()  
 print(f"Дизлайк для цитаты '{last\_quote}' увеличен.")  
  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Цитата '{last\_quote}' {action}нута.")  
  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при обновлении статуса цитаты.")  
  
# Функции удаления цитат:  
  
def delete\_quote(update: Update, context: CallbackContext):  
 if not context.args:  
 update.message.reply\_text("Укажите текст цитаты для удаления.")  
 return  
  
 to\_remove = ' '.join(context.args)  
 remove\_quote\_or\_category(update, context, to\_remove)  
  
def confirm\_deletion(update, context, to\_remove):  
 keyboard = [['Да', 'Нет']]  
 context.user\_data['delete\_quote'] = to\_remove # Сохраняем цитату для удаления в данных пользователя  
 update.message.reply\_text(  
 "Вы уверены, что хотите удалить цитату?",  
 reply\_markup=ReplyKeyboardMarkup(keyboard, one\_time\_keyboard=True),  
 )  
  
def remove\_quote\_or\_category(update: Update, context: CallbackContext, to\_remove: str, is\_category: bool = False):  
 if not to\_remove:  
 update.message.reply\_text(f"Укажите {'категорию' if is\_category else 'текст цитаты'} для удаления.")  
 return  
  
 removed = False # Флаг для отслеживания удаления  
  
 # Удаляем категорию  
 if is\_category:  
 if to\_remove.lower() in quote\_generator.categories:  
 del quote\_generator.categories[to\_remove.lower()]  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Категория '{to\_remove}' успешно удалена.")  
 else:  
 update.message.reply\_text(f"Категория '{to\_remove}' не найдена.")  
 else:  
 # Удаляем цитату из категорий  
 removed = False  
 for category, quotes in quote\_generator.categories.items():  
 updated\_quotes = quotes.copy() # Создаем копию списка цитат  
  
 updated\_quotes = [quote for quote in updated\_quotes if quote.text.lower() != to\_remove.lower()]  
  
 if len(updated\_quotes) < len(quotes):  
 removed = True  
 quote\_generator.categories[category] = updated\_quotes  
 quote\_texts = [quote.text for quote in updated\_quotes]  
 print(f"Категория: {category}, Цитаты: {quote\_texts}")  
  
 if not removed:  
 update.message.reply\_text(f"Цитата '{to\_remove}' не найдена в категориях.")  
 else:  
 quote\_generator.save\_quotes('quotes.json')  
 update.message.reply\_text(f"Цитата '{to\_remove}' успешно удалена из категорий.")  
  
 # Применение функции  
 remove\_quote\_or\_category(update, context, 'текст\_цитаты\_для\_удаления')  
  
  
# Функции анализа цитат:  
def analyze\_quotes(update, context):  
 try:  
 all\_quotes = [quote for quotes in quote\_generator.categories.values() for quote in quotes]  
 sorted\_quotes = sorted(all\_quotes, key=lambda x: x.likes / max(1, x.dislikes), reverse=True)  
  
 top\_5\_quotes = sorted\_quotes[:5]  
 message = "Топ 5 цитат:\n"  
 for index, quote in enumerate(top\_5\_quotes, start=1):  
 message += f"{index}. '{quote.text}' - Лайков: {quote.likes}, Дизлайков: {quote.dislikes}\n"  
  
 update.message.reply\_text(message)  
  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при анализе цитат.")  
  
  
  
# Функции управления командами:  
def start(update: Update, context: CallbackContext) -> None:  
 existing\_commands = ["/start", "/addquote", "/deletequote", "/deletecategory", "/category", "/resetlikes", "/analyze"]  
 all\_commands\_text = '\n'.join(existing\_commands)  
 update.message.reply\_text(f"Привет! Чтобы получить цитату, используйте команды:\n{all\_commands\_text}")  
def sanitize\_command\_name(command):  
 return ''.join(c.lower() for c in command if c.isalnum())  
  
def is\_valid\_command(command):  
 return bool(re.match(r'^[A-Za-z0-9\_]+$', command))  
  
from telegram import Update, ReplyKeyboardMarkup  
from telegram.ext import CallbackContext  
def reset\_likes\_and\_dislikes(update: Update, context: CallbackContext):  
 try:  
 if not context.bot.get\_chat\_member(update.message.chat\_id, update.message.from\_user.id).can\_promote\_members:  
 update.message.reply\_text("У вас нет прав для выполнения этой операции.")  
 return  
  
 quote\_generator.reset\_likes\_and\_dislikes()  
 quote\_generator.save\_quotes('quotes.json')  
  
 update.message.reply\_text("Лайки и дизлайки для всех цитат были обнулены.")  
  
 except Exception as e:  
 logger.error(f"Произошла ошибка: {e}")  
 update.message.reply\_text("Произошла ошибка при сбросе лайков и дизлайков.")  
  
  
  
  
  
  
  
  
  
  
def main() -> None:  
 updater = Updater("6819977499:AAEQyhjz4sYhJhasJLhyVk3II-LEzDs5fQQ")  
 dp = updater.dispatcher  
  
 # Обработчики команд  
 dp.add\_handler(CommandHandler("start", start))  
 dp.add\_handler(CommandHandler("addquote", add\_quote))  
 dp.add\_handler(CommandHandler("getquote", get\_quote))  
 dp.add\_handler(CommandHandler("category", category\_list))  
 dp.add\_handler(CommandHandler("deletecategory", delete\_category\_command))  
 dp.add\_handler(CommandHandler("resetlikes", reset\_likes\_and\_dislikes))  
 dp.add\_handler(CommandHandler("analyze", analyze\_quotes))  
 dp.add\_handler(CommandHandler("deletequote", delete\_quote))  
 dp.add\_handler(CommandHandler("like", like))  
 dp.add\_handler(CommandHandler("dislike", dislike))  
  
  
  
 quote\_generator.remove\_duplicate\_quotes('quotes.json') # Удаление повторяющихся цитат  
 quote\_generator.load\_quotes('quotes.json') # Загрузка цитат из файла  
  
 updater.start\_polling()  
 updater.idle()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()