**ВСР 2.5**

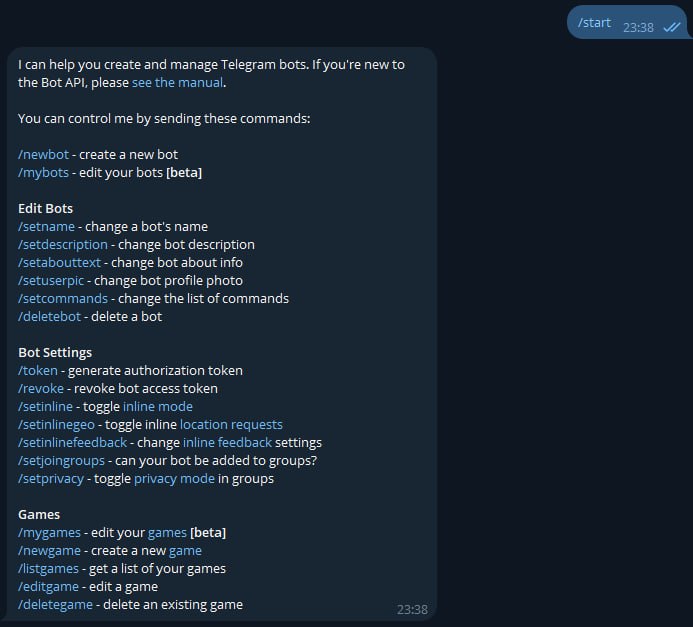
**(Создание бота для Telegram)**

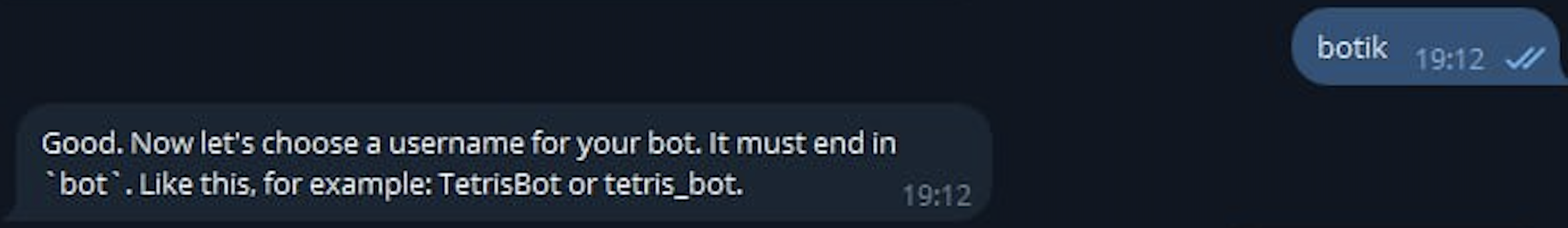
**Введение**

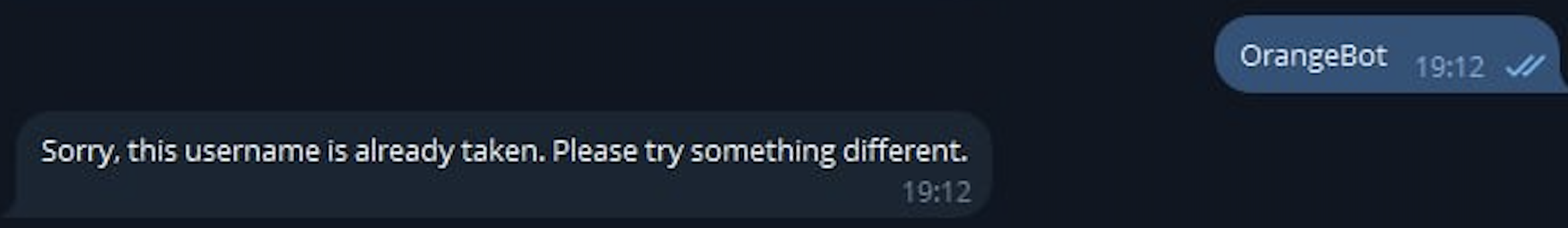
В рамках задания требовалось создать Telegram-бота с помощью онлайн-приложения. Однако в процессе работы было принято решение реализовать бота с использованием языка программирования Python, что потенциально позволяет добиться большей гибкости и расширенных возможностей.

В данном отчёте показан пошаговый аннотированный алгоритм создания Telegram-бота и написания для него кода на языке Python. Несмотря на отличие от изначально предложенного метода (онлайн-конструктор), принципы работы Telegram-ботов остаются неизменными:

**Регистрация бота**

После того, как была придумана идея и цель бота, в Telegram необходимо найти аккаунт BotFather, это бот, созданный командой разработчиков Telegram, который позволяет создавать и управлять ботами пользователей. После ввода команды «start», BotFather пришлёт список всех команд.

Сразу же можно нажать на команду /newbot, тем самым создав нового бота. После, BotFather вышлет сообщение, в котором попросит указать название этого бота.

Указав название, BotFather попросит придумать username.

Имя бота должно быть уникальным. Если оно используется каким-либо другим ботом, то BotFather скажет об этом и попросит придумать другое.

Когда будет введено имя, которое подходит под все критерии, BotFather выдаст специальный API-токен: это своего рода серийный номер, который понадобится в будущем для настройки и управления созданным ботом.

API-токен следует хранить в секрете, как и любой пароль, поэтому на картинке он скрыт.

**Программирование бота**

**Введение**

Python был выбран языком программирования на котором будет написан бот, потому что это удобный и мощный язык для которого существует огромное количество удобных библиотек, а также библиотек для работы с API Телеграма.

Для работы с API Телеграма была выбрана библиотека «**aiogram**», потому что она предоставляет удобный интерфейс для работы с функционалом Телеграм ботов. Также эта библиотека внутренне использует библиотеку «**asyncio**», которая позволяет писать асинхронный код. Благодаря этому, бота сможет использовать несколько человек одновременно.

1. **Инициализация бота**

Для начала необходимо инициализировать объект «**Bot**» из библиотеки «**aiogram**»:



Значением параметра «**token**» является API токен, который был получен от BotFather ранее.

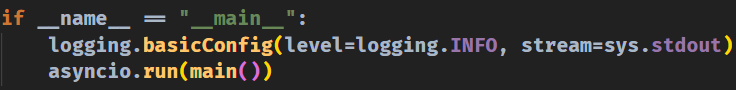
Затем нужно инициализировать объект «**Dispatcher**», он является главным «маршрутизатором» всех событий, которые будет получать бот (например получение сообщения от пользователя):



Далее в асинхронной функции «**main**» запускаем бота вызовом метода «**start\_polling**» у объекта «**Dispatcher**», передав в неё экземпляр объекта «**Bot**».



Затем необходимо инициализировать библиотеку логирования «**logging**» и вызвать асинхронную функцию «**main**» с помощью функции «**run**» из библиотеки «**asyncio**»:

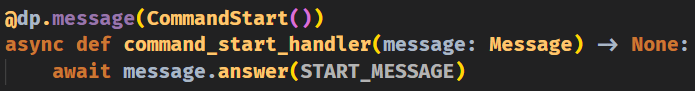


В консоль выведется информация об успешном запуске бота:



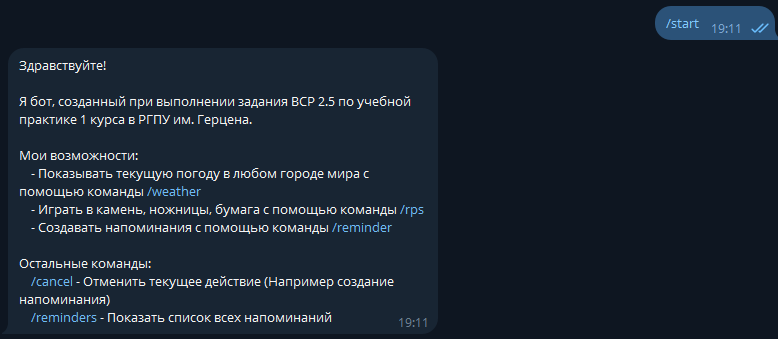
1. **Получение сообщений от пользователя**

Чтобы начать получать сообщения от пользователя, нужно «подписаться» на получение события «**message**». Для этого необходимо создать асинхронную функцию, которая будет вызываться при получении события. Затем необходимо связать эту функцию с событием «**message**» с помощью экземпляра объекта «**Dispatcher**»:



На картинке выше, мы создали асинхронную функцию «**command\_start\_handler**» с аргументом типа «**Message**» и подписали её на событие «**message**» с определённым фильтром «**CommandStart**». С фильтром «**CommandStart**» эта функция вызовется только тогда, когда пользователь отправит сообщение с командой «**start**».

В теле функции мы вызываем асинхронный метод «**answer**» у аргумента «**message**» типа «**Message**», передав в него текст приветственного сообщения. Вот как это будет выглядеть от лица пользователя:



1. **Реализация предоставления текущей погоды**

Наш бот должен уметь предоставлять пользователю текущую погоду в любом городе мира. Для этого мы будем использовать сервис «**OpenWeatherMap**», который позволяет бесплатно в небольших количествах использовать их API.

Для удобства, мы будем использовать библиотеку «**pyowm**», которая облегчит взаимодействие с API сервиса «**OpenWeatherMap**».

Сперва инициализируем объект «**OWM**» из библиотеки «**pyowm**»:

****

Первым аргументом конструктора объекта «**OWM**» является API токен сервиса «**OpenWeatherMap**», который можно получить, зарегистрировавшись на их официальном сайте.

Затем инициализируем объект «**WeatherManager**»,вызвав метод «**weather\_manager**» у созданного выше экземпляра объекта «**OWM**».

Далее создадим функцию для обработки события команды «/weather»:



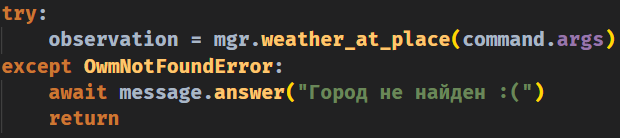
В начале тела функции «**command\_weather\_handler**» сделаем обработку случая, когда пользователь ввёл команду без указания города.



В случае, когда пользователь указал город сделаем запрос к сервису «**OpenWeatherMap**» для получения текущей погоды в этом городе:



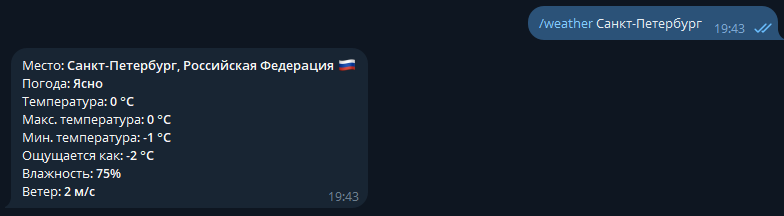
В случае, если указанный пользователем город не будет найден (например, если введённое название города не существует), необходимо уведомить его об этом:

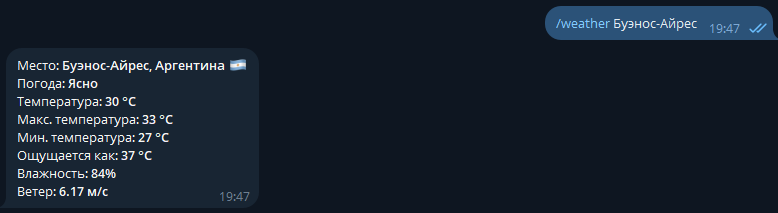


Если всё прошло успешно, отправим пользователю сообщение с информацией о погоде, текущей температуре, минимальной температуре, максимальной температуре, эффективной температуре (температура, которую ощущает человек при заданной скорости ветра и влажности воздуха), влажности и скорости ветра:



Вот как это выглядит от лица пользователя:

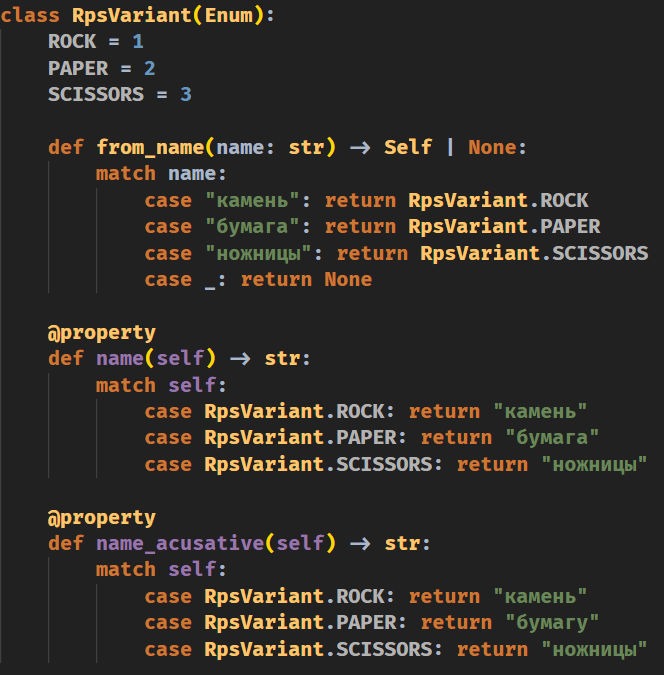






1. **Реализация игры «камень, ножницы, бумага»**

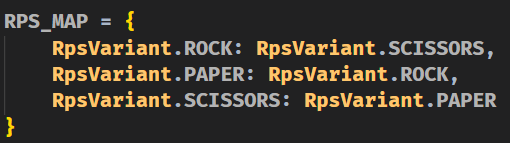
Научим нашего бота играть в игру «камень, ножницы, бумага». Для этого создадим класс «**RpsVariant**», который будет означать вариант ответа в игре:



Затем объявим список со всеми возможными вариантами в игре (он понадобиться нам позже, для случайного выбора ботом варианта):



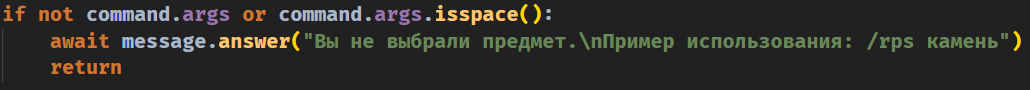
Далее объявим словарь, где в качестве ключа будет предмет, который выигрывает тот предмет, что является значением ключа:



Теперь объявим функцию для обработки события ввода команды «**rps**» пользователем:



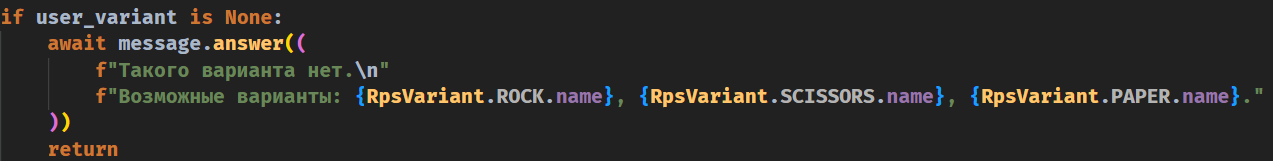
Далее в начале тела функции добавим обработку случая, когда пользователь на указал никакой предмет:



Затем преобразуем аргумент пользователя в экземпляр класса «**RpsVariant**»:



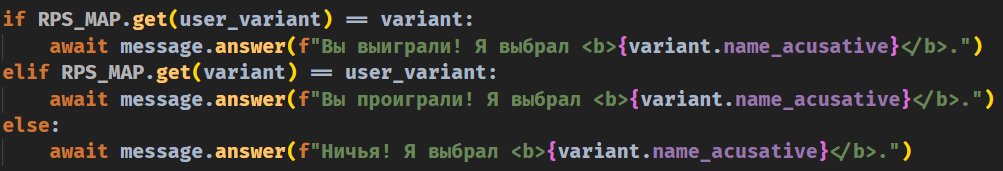
Добавим обработку случая, когда пользователь неправильно ввёл название предмета:



Случайно выберем предмет из списка вариантов в качестве выбора бота:



Затем с помощью словаря «**RPS\_MAP**» сообщим пользователю кто победил:



1. **Реализация напоминаний**

Наш бот должен уметь создавать напоминания и отправлять уведомления пользователям по истечении срока напоминания. Для этого нам понадобиться библиотека для работы с базами данных «**sqlite3**».

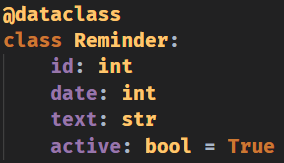
Создадим функцию для создания базы данных напоминаний для каждого пользователя:



Добавим вызов функции создания базы данных напоминаний в функцию обработки команды «**start**». Вызов этой функции должен осуществляться на другом потоке, для этого воспользуемся функцией «**to\_thread**» из библиотеки «**asyncio**»:



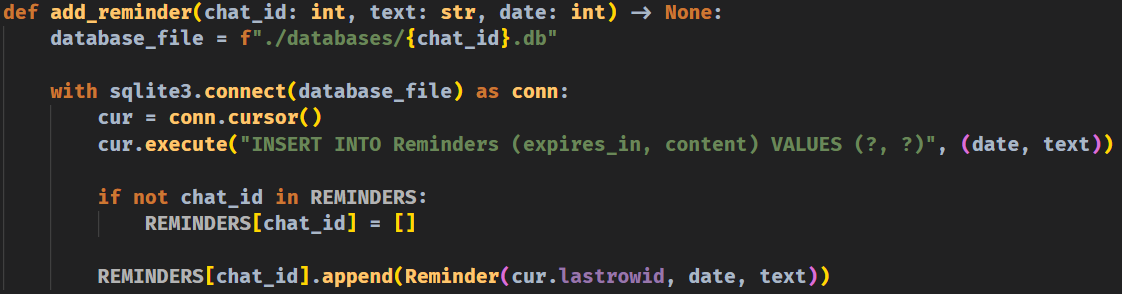
Далее объявим класс «Reminder», который будет хранить информацию об напоминании:



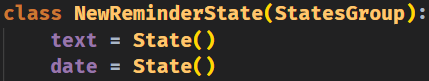
Далее создадим словарь, в который будут записываться напоминания всех пользователей (это нужно, чтобы уменьшить количество запросов к базе данных, тем самым ускорив работу бота):



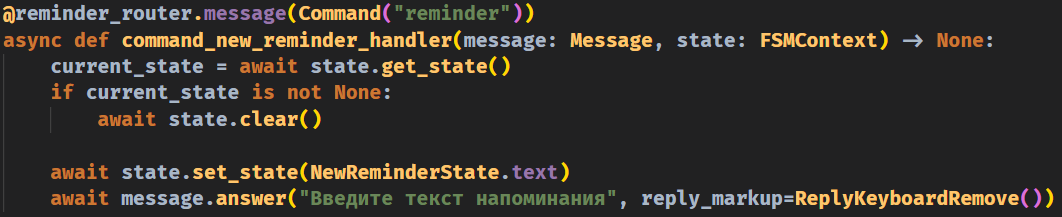
Создадим функцию для создания напоминания в базе данных:



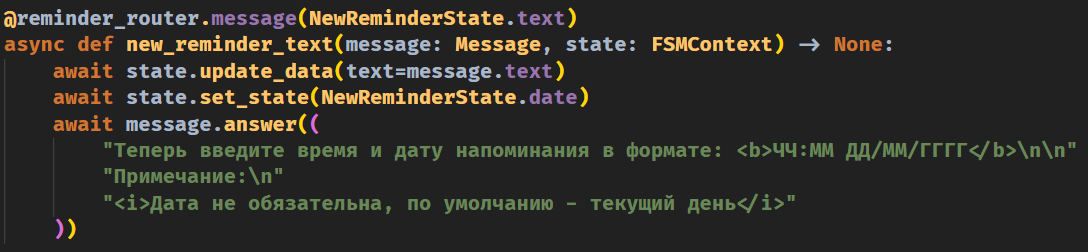
Далее объявим класс «**NewReminderState**», который будет хранить состояние диалога создания нового напоминания с пользователем:



Далее создадим функцию для обработки ввода команды «/reminder» пользователем:



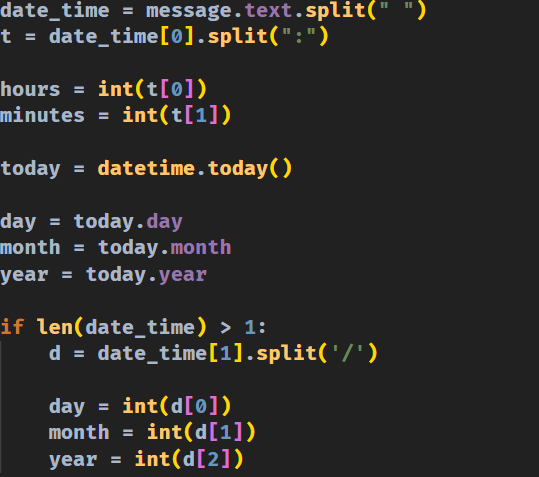
Затем создадим функцию для сохранения введённого пользователем текста напоминания:



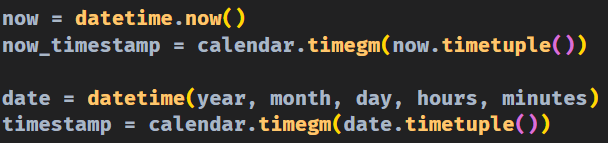
Теперь создадим функции для сохранения введённой пользователем даты напоминания:



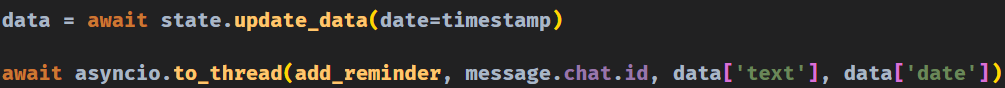
Разобьём введённую пользователем время и дату напоминания в часы, минуты, год, месяц и день, и преобразуем их в числа (код не содержит проверку даты на ошибки для экономии места):



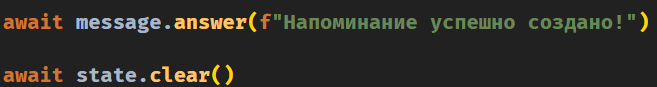
Далее получим текущую дату и преобразуем её и дату напоминания в формат «Unix-время»:



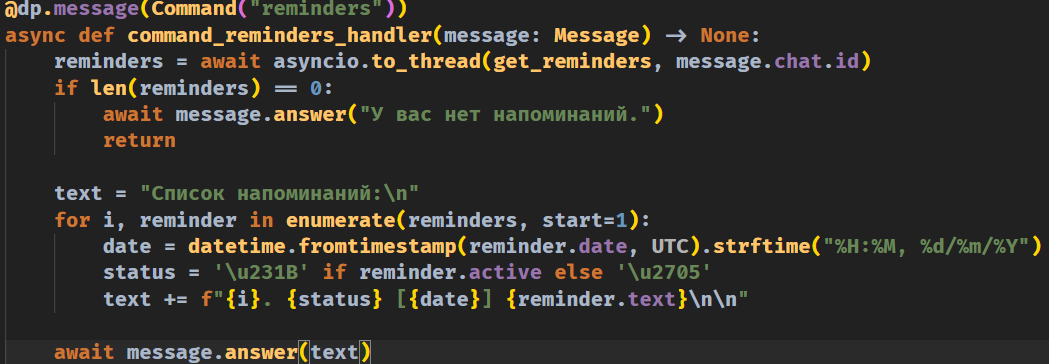
Затем добавим напоминание в базу данных, вызвав функцию «**add\_reminder**»:



Далее сообщим пользователю об успешном создании напоминания и очистим состояние диалога создания напоминания:



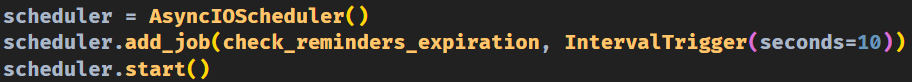
Создадим функцию для обработки ввода пользователем команды «**reminders**»:



Далее создадим функцию, которая будет отправлять напоминания пользователям при истечении времени.



Затем с помощью библиотеки «**apscheduler**», будем асинхронно вызывать эту функцию каждые 10 секунд:



**Заключение**

В данном отчёте были показаны наиболее важные этапы создания Telegram-бота, а также прокомментированы фрагменты его исходного кода. Благодаря коду на языке Python, в дальнейшем бота можно будет более гибко настраивать, добавлять новые функции, тем самым улучшая его функционал