TG02. Обработка входящих сообщений от пользователей и отправка текста и мультимедийных файлов

Предыдущий урок (/pl/teach/control/lesson/view?id=336393177)

Дорогой студент! Интерфейсы современных приложений могут очень быстро меняться. Если вдруг то, что видишь ты, отличается от того, что демонстрирует эксперт, настолько, что тебе непонятно, куда нажимать — напиши об этом в учебный чат, указав номер урока и видео. Наш куратор обязательно поможет тебе разобраться, а мы постараемся как можно скорее актуализировать материал :)

Что мы умеем:

✅ Изучили фреймворк Flask
✅ Изучили фреймворк Django
✅ Изучили процесс установки и настройки aiogram

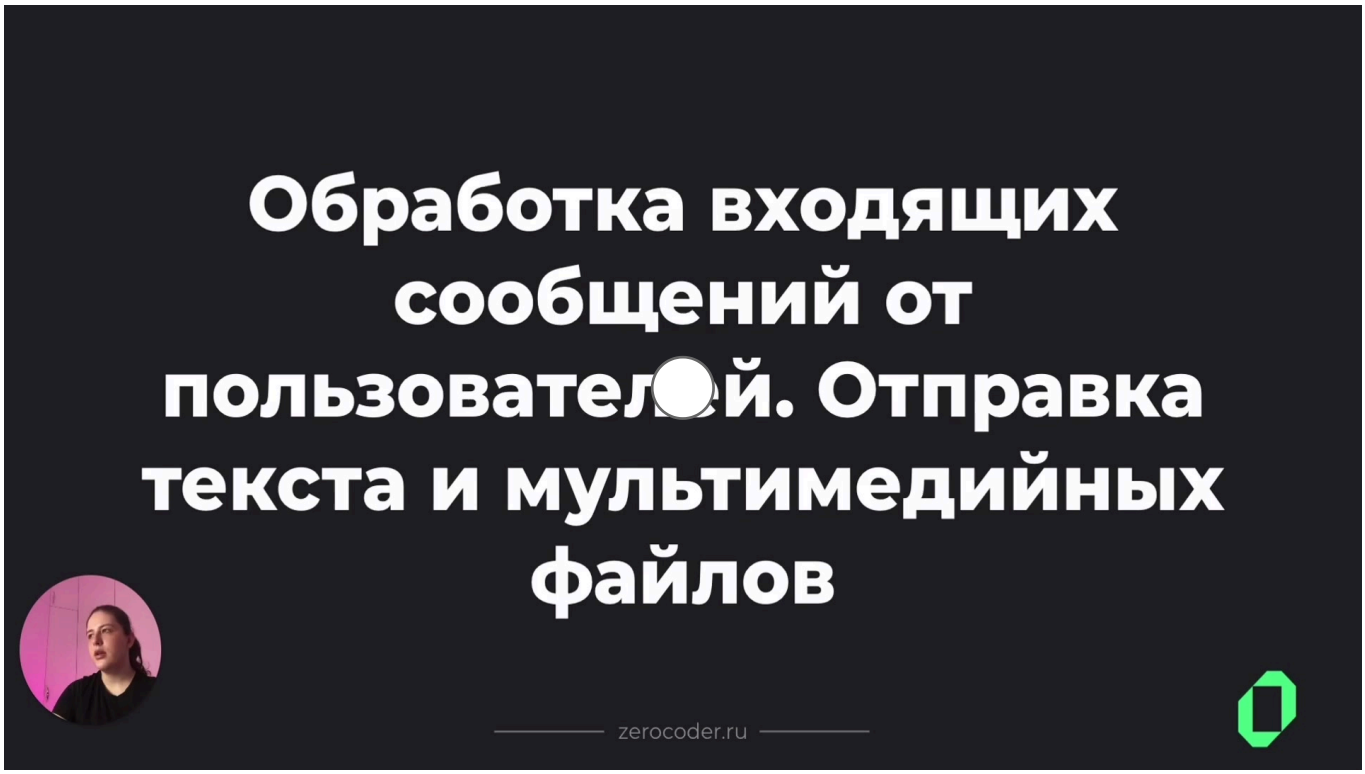
Результат дня

Разработаем три новые функции для Telegram-бота, закрепим знания с прошлого урока и познакомимся с новыми понятиями и возможностями для обработки входящих сообщений и отправки текста и мультимедийных файлов.

▶️ Время чтения и просмотра: ~ 1 час

Теория на сегодня:

1. План урока. Настройка универсального обработчика сообщений



Мы продолжаем говорить об особенностях работы с Telegram-ботами. Сегодняшняя наша тема — это обработка входящих сообщений от пользователей, а также отправка текста и мультимедийных файлов.

План урока таков:

- Мы разработаем три новые функции для бота;
- Закрепим знания с прошлого урока;
- Познакомимся с новыми понятиями и возможностями бота.

Всё это мы будем отрабатывать на практике, поэтому сразу же открываем PyCharm.

В прошлом уроке мы рассмотрели возможность работы с командами и с получением сообщений. Мы получали текст, изображения, а также создавали те или иные команды.

Теперь поговорим о том, как бот будет реагировать на любые сообщения, которые ему отправляет пользователь.

Настройка универсального обработчика сообщений

1. Открываем код из прошлого урока и запускаем его.
2. Переходим в Telegram-бота и пишем ему любой текст, например: “Привет”. Сейчас он никак не будет реагировать на эти сообщения: они доставлены и прочитаны, но дальше ничего не происходит, потому что мы не прописали обработку рандомных сообщений.
3. Для этого мы пропишем небольшую функцию для нашего Telegram-бота, чтобы мы могли обрабатывать любые сообщения, которые поступают в бот.
4. Чтобы создать функцию, которая обрабатывает все входящие сообщения, нужно прописать пустой декоратор. Мы не будем прописывать в нем никакую команду и никакой текст. Прописываем:

   ```
   @dp.message()
   async def start(message: Message):
       await message.answer("Приветики, я бот!")
   ```

   Таким образом, мы получаем обычный стандартный декоратор. Декораторы, прописанные выше, срабатывают только при поступлении какой-либо команды: help, start или определенного текста. При создании пустого декоратора будет обрабатываться любой текст, какой только может быть.

   Сейчас, на любой текст, который пользователь отправит, будет приходить сообщение «Приветики, я бот», так как это указано в нашем декораторе.

5. Вместо этого пропишем что-нибудь другое, например “Я тебе ответил”:

```
@dp.message()
async def start(message: Message):
    await message.answer("Я тебе ответил")
```

6. Запускаем код заново и смотрим на результат в Telegram-боте. Пишем ему снова "привет", и он отвечает "Я тебе ответил".

Таким образом, сообщения, поступающие боту, не относящиеся ни к каким командам, будут обрабатывать отдельно, на них всегда будет поступать какой-то определенный ответ.

Создание эхо-бота

Эхо-бот — это один из самых простых и популярных ботов, которых только создают. Эхо-бот работает по принципу возвращения того же сообщения, которое ему отправил пользователь. Попробуем создать такого эхо-бота.

1. Прописываем в декораторе send_copy, что означает, что мы будем отправлять пользователю копию сообщения:

```
@dp.message()
async def start(message: Message):
    await message.send_copy(chat_id=message.chat.id)
```

2. Теперь на все сообщения, которые будут приходить, бот будет отвечать тем же сообщением. Перезапускаем код и переходим в Telegram для проверки бота. Пишем ему "привет", и он отвечает тем же. Пишем что-то другое и получаем такое же сообщение. При этом команды в прежнем режиме.

> Важно прописывать декоратор копирования всех сообщений именно внизу кода. Если этот элемент кода будет стоять выше, то при отправке в бот, например, команды start бот просто переотправит в ответ “start” вместо выполнения команды. Другими словами, эта команда не успевает дойти до обработчика и перехватывается быстрее данным декоратором.

Работа с информацией о пользователе

Рассмотрим работу с информацией о пользователе. Для этого немного видоизменим команду start.

1. Вместо "я бот" напишем "приветики" и укажем имя пользователя:

```
@dp.message_handler(CommandStart())
async def start(message: Message):
    await message.answer(f'Приветики, {message.from_user.first_name}')
```

Также можно использовать полное имя (в приветствии будут указываться и имя, и фамилия):

```
f'Приветики, {message.from_user.full_name}'
```

2. Перезапускаем код и вновь проверяем бота: отправляем ему команду **/start**, и в ответ получаем приветствие с указанием имени пользователя.

На самом деле в переменной `message` хранится много информации, не только имя пользователя, но и эта информация в том числе, что позволяет нам работать с этими данными.

Работа с префиксами

Ещё одна важная часть — работа с префиксами. Когда мы отправляем любую команду боту в Telegram, по умолчанию используется слэш перед командой. Но мы можем задать и свои префиксы.

1. Например, зададим еще какой-нибудь префикс для фото:

```
@bot.message_handler(commands('photos', prefix='&'))
```

2. Перезапускаем код и проверяем работу в Telegram-боте: прописываем **&photo** и получаем в ответ изображение. В качестве префикса можно прописать что угодно.

Скачивание файлов

Мы также можем скачивать то, что в бот отправляет пользователь, например, изображения.

1. У нас в коде уже есть фрагмент с обработкой фотографий. Мы продолжим работать с этим фрагментом, дописав его:

```
@dp.message(F.photo)
async def react_photo(message: Message):
                list = ['Ого, какая фотка!', 'Непонятно, что это такое', 'Не отправляй мне такое больше']
    rand_answ = random.choice(list)
    await message.answer(rand_answ)
                await bot.download(message.photo[-1],destination=f'tmp/{message.photo[-1].file_id}.jpg')
```

Рассмотрим этот код чуть подробнее. В круглых скобках мы указали несколько важных параметров. Сначала указываем message.photo и в квадратных скобках -**1**. Затем добавляем запятую и указываем атрибут destination, который равен строке 'tmp/'. Это будет наша папка для хранения файлов. В фигурных скобках мы прописали message.photo[-1].file_id. Это делается для того, чтобы у фотографий были уникальные имена. Таким образом, мы будем сохранять все фотографии в папке **tmp**, и у каждой фотографии будет название, соответствующее её ID.

Почему мы используем **-1** в квадратных скобках? Когда мы отправляем фотографию, Telegram отправляет несколько ее копий в разных размерах. Мы выбираем последнюю версию, так как она имеет максимальный доступный размер и наиболее удобна для нас. Поэтому указываем **-1**, что означает последнюю версию изображения. И, наконец, мы указываем ID фотографии, чтобы присвоить ей уникальное имя.

2. Создаем в проекте папку **tmp**, чтобы можно было с ней работать (**New** — **Directory** — и прописываем название папки: “**tmp**”).
3. Перезапускаем код. Переходим в Telegram-бот и отправляем ему какое-нибудь изображение. Возвращаемся в PyCharm и проверяем: теперь в папке **tmp** сохранена только что отправленная фотография, при этом в качестве названия он использовал id изображения.

Отслеживание сообщений с помощью условного оператора

Для отслеживания определенных сообщений с текстом можно также использовать условия.

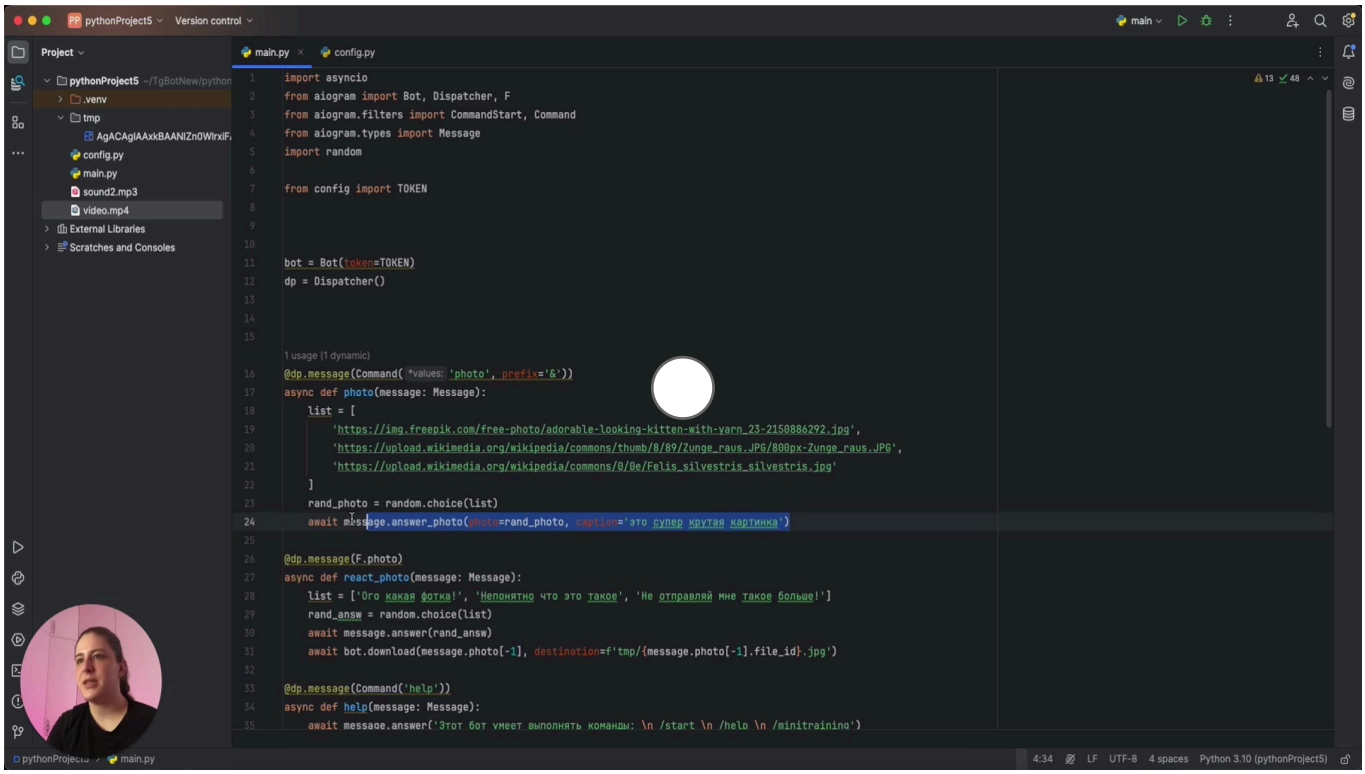
1. Прописываем:

```
@dp.message()
async.def start(message: Message)
                if message.text.lower() == 'test':
                                await message.answer('Тестируем')
```

Рассмотрим этот код. Для начала мы объявляем декоратор @dp.message(), который указывает, что функция будет обрабатывать сообщения. Затем объявляем асинхронную функцию `start`, которая принимает параметр `message` типа **Message**. Внутри функции проверяем, равно ли текст сообщения **message.text** строке 'test' с приведением к нижнему регистру (lower). Это делается для того, чтобы условие срабатывало независимо от регистра введенного текста.

2. Снова запускаем код и тестируем Telegram-бота: пишем ему слово “test”, и он отвечает: “Тестируем”. Это еще один способ отслеживать текст.

2. Работа с мультимедийными файлами в Telegram-боте



Отправка видео и аудио

Теперь рассмотрим, как отправлять различные файлы через нашего бота. На прошлом уроке мы изучали отправку фотограф добавим к этому функционалу отправку видео и аудио. Для этого создадим соответствующие функции.

1. Сначала создадим команду и функцию для отправки видео:

```
@dp.message(Command('video'))
async def video(message: Message):
                await message.answer("Этот бот умеет выполнять команды:\\n/start\\n/help\\n/minitraining")
```

2. Создадим также команду для отправки аудио:

```
@dp.message(Command('audio'))
async def audio(message: Message):
    await message.answer("Этот бот умеет выполнять команды:\\n/start\\n/help\\n/minitraining")
```

3. Для работы с файлами нужно импортировать класс `FSInputFile`. Он необходим для обработки файлов и их отправки в а строке, с помощью которой мы уже импортировали aiogram, дописываем `FSInputFile`, таким образом получая строку:

```
from aiogram.types import Message, FSInputFile
```

4. На следующем шаге мы создадим переменную, в которой будет храниться объект класса, который мы только что импорт этого прописываем:

```
video = FSInputFile('video.mp4')
```

В этом примере мы создаем переменную `video`, в которой хранится объект класса `FSInputFile`. В скобках указываем пу Если файл находится в одной папке с `main.py`, достаточно указать только его название. Если он находится в другой паг названием нужно прописать полностью путь к файлу.

5. Теперь используем две команды для отправки видео: `message.answerVideo` или `bot.sendVideo`. Также указываем ID чата, с команда, и переменную `video`.

```
await bot.send_video(message.chat.id, video)
```

Действия

6. Запускаем код и снова идем в Telegram-бота. Если теперь ввести команду “Видео” в боте, в ответ должно отправиться в
7. Для отправки тяжелых файлов, процесс может занять время. Чтобы пользователь видел, что бот не завис, можно пропис уведомление о загрузке. Для этого вернемся к фрагменту кода для отработки видео и после вызова асинхронной функ

```
await bot.send_chat_action(message.chat.id, 'upload_video')
```

   Это уведомление покажет, что бот загружает видео. Когда загрузка завершится, уведомление исчезнет.
8. Перезапускаем код, заходим в Telegram-бота и отправляем ему команду /video, на что бот сообщает о том, что видео от (sending video).
9. Добавим команду для отправки аудио. В проекте заранее должен быть подготовлен аудиофайл. Создаем переменную которой сохраним объект класса `FSInputFile`. В круглых скобках указываем название аудиофайла:

```
audio = FSInputFile('sound2.mp3')
await bot.send_audio(message.chat.id, audio)
```

10. Снова запускаем код и проверяем работу в Telegram-боте: отправляем в чат команду /audio и в ответ получаем аудиоф

Озвучивание текста

1. Попробуем использовать всё сделанное выше, чтобы создать какую-нибудь необычную функцию — например, функцию тренировок, которая отправлялась бы отдельным файлом. При этом в квадратных скобках укажем тренировки, которые сгенерировать ChatGPT. Тренировки будут выбираться рандомным образом и сопровождаться небольшим текстом: “Это тренировка на сегодня”:

```
@dp.message(Command('training'))
async def training(message: Message):
   training_list = [
       "Тренировка 1:\\n1. Скручивания: 3 подхода по 15 повторений\\n2. Велосипед: 3 подхода по 20 повторений (каждая
       "Тренировка 2:\\n1. Подъемы ног: 3 подхода по 15 повторений\\n2. Русский твист: 3 подхода по 20 повторений (каж
       "Тренировка 3:\\n1. Скручивания с поднятыми ногами: 3 подхода по 15 повторений\\n2. Горизонтальные ножницы: 3 по
   ]
   rand_tr = random.choice(training_list)
   await message.answer(f"Это ваша мини-тренировка на сегодня {rand_tr}")
```

2. Также нужно сделать озвучку. Для этого понадобится специальная библиотека Google TTS (мы уже с ней работали в пр уроках).
3. Заходим в Python Packages, в строке поиска пишем “gtts” и нажимаем **`Install packsges`**. Когда загрузка будет завершен импортируем нужные библиотеки:

```
from gtts import gTTS
import os
```

4. Создадим объект класса GoogleTTS. В круглых скобках при этом указываем параметр test и задаем ему значение, какой нужно преобразовать в голос. Этот текст хранится в переменной rand_tr. Также необходимо указать, на каком языке на озвучка, так как по умолчанию это будет английский язык:

```
tts = gTTS(text=rand_tr, lang='ru')
```

5. Чтобы сохранить то, что мы создадим, нужно прописать специальную команду `save`, указав при этом название файла, в будут сохраняться эти данные

```
tts.save("training.mp3")
```

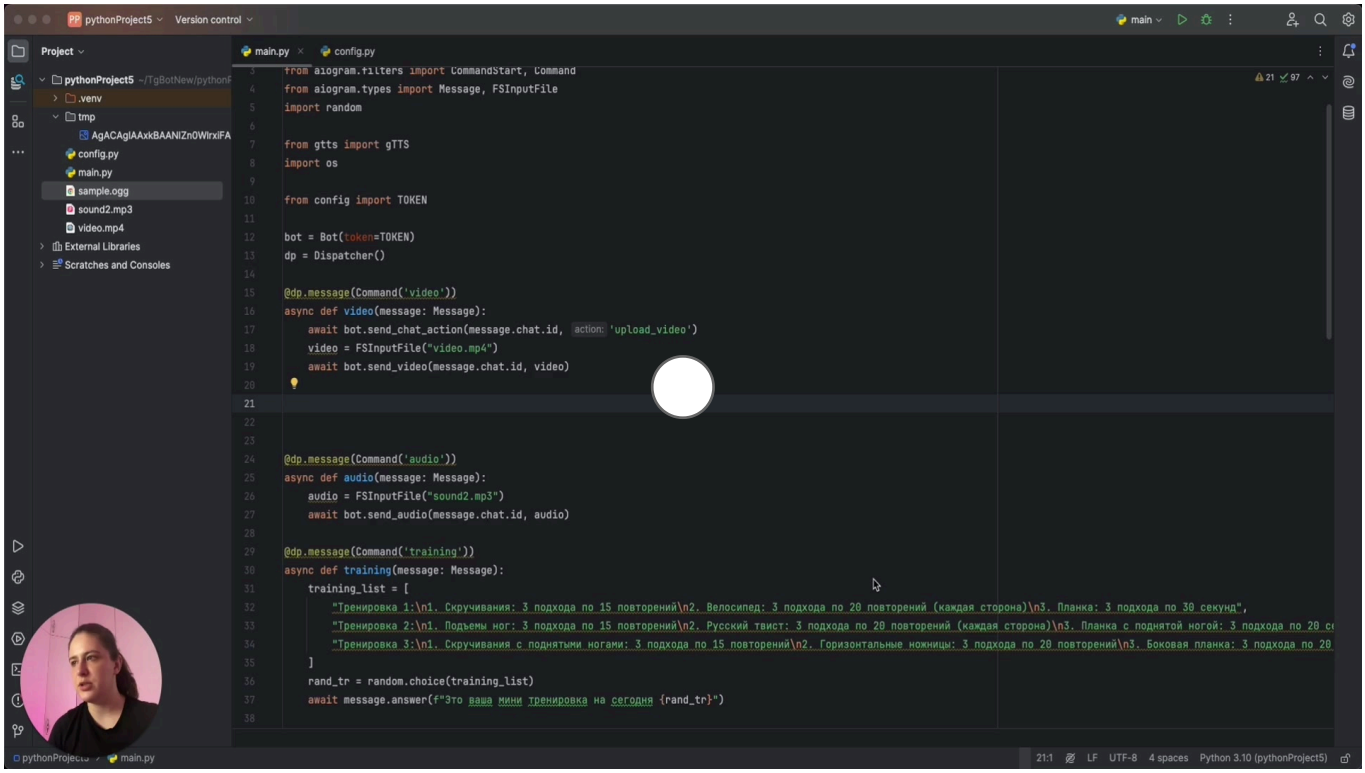
Действия

6. Далее эти данные нужно отправить. Для этого создаем переменную audio, в круглых скобках прописываем название фа отправляем — для этого мы используем `await`, а в круглых скобках указываем **** `chatid` и передаем аудио для отправк прописываем `remove`, так как после отправки файл можно удалить:

```
audio = FSInputFile('training.mp3')
await bot.send_audio(message.chat.id, audio)
os.remove("training.mp3")
```

7. Запускаем код и проверяем работу Telegram-бота. В чате с ботом прописываем команду /training. В ответ нам приходит тренировки, а также озвучивание этой тренировки.

3. Работа с голосовыми сообщениями и документами в Telegram-боте. Итоги урока



Отправка голосовых сообщений

В этой части урока мы будем работать с голосовыми сообщениями, поскольку этот формат немного отличается от формата аудиофайла. Аудиофайл можно скачать; скачать голосовые сообщения тоже можно, но это не так просто.

1. Для начала создадим еще одну функцию для примера. Назовем команду и функцию "voice". Что важно знать о голосовых сообщениях: формат MP3 здесь не поддерживается, для этого нужен формат OGG. Сначала попробуем просто отправить голосовое сообщение, а затем рассмотрим, что еще можно сделать.

```
@dp.message(Command('voice'))
async def voice(message: Message):
```

2. Теперь пропишем, как работать с голосовым сообщением. Для этого нам также нужен FSInputFile. Создаем переменную `voice`, далее указываем `FSInputFile` и в круглых скобках указываем файл, который будем отправлять в формате OGG. десь мы также можем использовать `message.answer` — в этом случае не нужно будет указывать `chat id`.

```
voice = FSInputFile("sample.ogg")
await message.answer_voice(voice)
```

Действия

3. Запускаем программу, вызываем команду /voice и получаем в ответ голосовое сообщение. Проверим: звук гудка работает.
4. Теперь применим этот формат для отправки нашей тренировки. Текущий формат аудиозаписи тренировки — это файл формата MP3, который можно загрузить и скачать. Попробуем отправить ту же самую информацию голосовым сообщением. Для этого нам нужно будет просто поменять формат сохранения.
5. Обратимся к фрагменту кода, где мы работали с библиотекой gTTS. В этом фрагменте прописано, что озвученный текст должен сохраняться в формат mp3. Вместо этого пропишем везде вместо mp3 формат ogg:

```
tts = gtts(text=rand_training, lang='ru')
tts.save("training.ogg")
audio = FSInputFile("training.ogg")
await bot.send_voice(chat_id=message.chat.id, voice=audio)
os.remove("training.ogg")
```

6. Запускаем код и проверяем команду в Telegram-боте: прописываем команду /training. В ответ мы получаем в чате озвучку тренировки, отправленную в виде голосового сообщения.

Отправка документов

Теперь посмотрим, как отправлять документы. Это делается почти так же, но с другой функцией — функцией `doc` .

1. Прописываем:

```
@dp.message(Command('doc'))
                async def doc(message: Message):
                doc = FSInputFile("TG02.pdf")
                await bot.send_document(message.chat.id, doc)
```

2. Здесь мы создали функцию и команду **doc**, переменную `doc` и указали название PDF-файла, который будем отправлять. Чтобы отправить этот файл, используем команду `bot.send_document` , указываем `chat.id` и переменную `doc` .
3. Запускаем программу, вызываем команду /doc, и нам приходит указанный PDF-файл.

Итоги урока

На этом наш сегодняшний урок завершен. Подведем итоги.

- Мы разработали новые функции для бота;
- Закрепили знания с прошлого урока;
- Познакомились с новыми понятиями и возможностями бота.

Время выполнить задание

1. Напишите код для сохранения всех фото, которые отправляет пользователь боту в папке img
2. Отправьте с помощью бота голосовое сообщение
3. Напишите код для перевода любого текста, который пишет пользователь боту, на английский язык

На следующем занятии

Будем работать с базами данных и свяжем их с Telegram-ботом, чтобы добавить ему новый функционал.

Если у вас есть вопросы, задавайте их, пожалуйста, в учебном чате, а не в этой анкете, — так вы сможете быстрее получить ответ.

TG02. Обработка входящих сообщений от пользователей и отправка текста и мультимедийных файлов

Понравился ли урок?

- ○ Понравился, все было понятно
- ○ Неплохо, но не все получилось/не все понятно
- ○ Не понравился

Обратная связь по уроку (не обязательно)

Отправить

ЗАДАНИЕ

Напишите код для сохранения всех фото, которые отправляет пользователь боту в папке img
Отправьте с помощью бота голосовое сообщение
Напишите код для перевода любого текста, который пишет пользователь боту, на английский язык

Ваш ответ

Прикрепить файлы

максимальный размер файла - 100мб

Отправить ответ **Сохранить как черновик**

20 монет за выполнение этого урока

Действия