

Создание и запуск чат-бота

Конспект занятия 2

Сервисы Telegram и другие упоминаются только в образовательных целях и не являются рекламой.

1. Цели занятия

- **Научиться обрабатывать несколько различных команд** (помимо `/start`): например, `/help`, `/info`, `/setname`, `/greet`.
- **Освоить работу с интерактивными элементами** (Reply-клавиатуры и Inline-кнопки), чтобы пользователь мог легко взаимодействовать с ботом, нажимая на готовые кнопки вместо ввода команд вручную.
- **Научиться сохранять пользовательские данные** (например, имя) в памяти бота или базе данных, чтобы бот «помнил» информацию и мог отвечать более персонализировано.
- **Дополнительно:** научиться узнавать `file_id` стикеров и отправлять их в сообщениях.

2. Обработка команд и интерактивность

Подробное объяснение вы найдёте в [мини-учебнике](#).

2.1. Обработка нескольких команд

Ранее вы уже видели пример команды `/start`. Теперь мы напишем ещё 2 команды: `/help` и `/info`.

Почему важно, чтобы у бота было несколько команд?

Команды упрощают взаимодействие пользователя с ботом: человек видит список доступных функций и понимает, как их вызвать.

Например, `/help` всегда полезна для подсказок.

Пример кода (асинхронный подход, v.20+ библиотеки python-telegram-bot):

```
from telegram import Update
from telegram.ext import CommandHandler, ContextTypes

# Команда /help подсказывает, что может делать бот
async def help_command(update: Update, context: ContextTypes.
DEFAULT_TYPE):
    await update.message.reply_text(
        "Я простой бот. Напиши /info, чтобы узнать обо мне
побольше."
    )

# Команда /info даёт информацию о боте
async def info_command(update: Update, context: ContextTypes.
DEFAULT_TYPE):
    await update.message.reply_text(
        "Я создан, чтобы демонстрировать возможности python-
telegram-bot. Попробуй написать что-нибудь!"
    )
```

Регистрация обработчиков в основном файле бота (например, в функции `main()`):

```
application.add_handler(CommandHandler("help", help_command))
application.add_handler(CommandHandler("info", info_command))
```

Проверка:

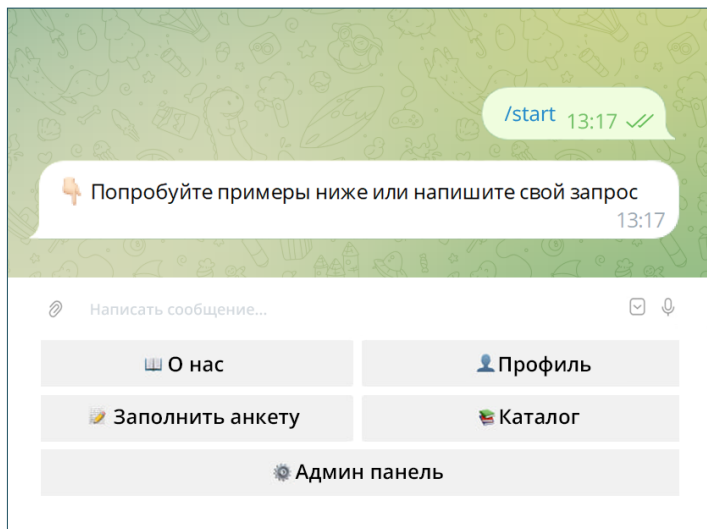
1. Запустите бота в Telegram.
2. Отправьте ему `/help` и `/info`.
3. Убедитесь, что бот правильно отвечает. Если есть проблемы, проверьте ещё раз код и регистрацию обработчиков.

2.2. Интерактивные клавиатуры

Помимо команд, в Telegram можно использовать **2 вида кнопок**, которые значительно упрощают жизнь пользователям:

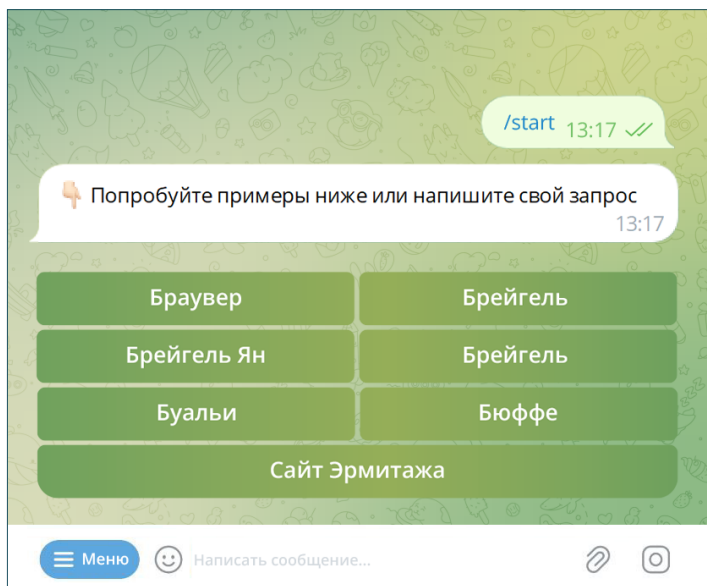
- **Reply-клавиатура (ReplyKeyboardMarkup)**

Отображается под полем ввода сообщения, заменяя стандартную клавиатуру телефона (или компьютера).



- **Inline-кнопки (InlineKeyboardMarkup)**

Встраиваются внутрь сообщений. При нажатии на них либо открывается ссылка, либо происходит действие (по `callback_data`) внутри бота.



2.2.1. Reply-клавиатура

Предположим, мы хотим, чтобы при вводе `/start` бот показывал две кнопки: «Привет!» и «Как дела?».

```
from telegram import ReplyKeyboardMarkup, KeyboardButton

async def start_command(update: Update, context: ContextTypes.
DEFAULT_TYPE):
    # Создаём двухкнопочную клавиатуру
    keyboard = [
        [KeyboardButton("Привет!"),
         KeyboardButton("Как дела?")]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_
keyboard=True)

    await update.message.reply_text(
        "Привет! Нажми на кнопку:",
        reply_markup=reply_markup
    )
```

- `resize_keyboard=True` позволяет клавиатуре автоматически подстраиваться под содержимое.
- Пользователь увидит кнопки вместо обычной клавиатуры и сможет нажать одну из них.

2.2.2. Inline-кнопки

Inline-кнопки появляются внутри сообщения. Например, при вызове `/help` мы можем добавить кнопку «Перейти на сайт»:

```
from telegram import InlineKeyboardMarkup, InlineKeyboardButton

async def help_command(update: Update, context: ContextTypes.
DEFAULT_TYPE):
    keyboard = [
        [InlineKeyboardButton("Перейти на сайт", url="https://
example.com")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)

    await update.message.reply_text(
        "Нажми на кнопку, чтобы открыть сайт:",
        reply_markup=reply_markup
    )
```

- Чтобы обработать такое нажатие, понадобится `CallbackQueryHandler`, о котором можно узнать в документации.

3. Работа с пользовательскими данными

Подробное объяснение вы найдёте в [мини-учебнике](#).

Можете изучить раздел 11 в учебнике, чтобы узнать, как сохранять данные в файле и работать с ними даже после перезапуска.

Иногда боту нужно «помнить» некоторую информацию о пользователе. Например, при вводе команды `/setname Вася` бот должен сохранить, что данного пользователя (с Telegram ID 1234) зовут Вася, и при вызове `/greet` говорить: «Привет, Вася!»

3.1. Сохранение данных в памяти (словарь)

Самый простой способ — создать словарь в файле бота:

```
ser_data = {} # Будет хранить пары {user_id: name}
```

Команда `/setname`

```
async def setname_command(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    # /setname Вася
    args = context.args # список слов, идущих после /setname
    if args:
        name = " ".join(args)
        user_id = update.effective_user.id
        user_data[user_id] = name
        await update.message.reply_text(f"Отлично, {name}! Те-
перь я тебя запомнил.")
    else:
        await update.message.reply_text("Пожалуйста, укажи своё
имя после команды /setname")
```

Команда /greet

```
async def greet_command(update: Update, context: ContextTypes.  
DEFAULT_TYPE):  
    user_id = update.effective_user.id  
    if user_id in user_data:  
        name = user_data[user_id]  
        await update.message.reply_text(f"Привет, {name}! Чем я  
могу помочь?")  
    else:  
        await update.message.reply_text("Я не знаю твоего име-  
ни. Используй /setname <имя>.")
```

Регистрация

```
application.  
add_handler(CommandHandler("setname", setname_command))  
application.  
add_handler(CommandHandler("greet", greet_command))
```

Плюсы и минусы словаря

- Плюсы: быстро и просто, минимальный код.
- Минусы: информация стирается при перезапуске бота.

3.2. Сохранение в базе данных (опционально)

Чтобы бот «помнил» данные между перезапусками, можно использовать [SQLite](#) – лёгкую встраиваемую базу данных.

Создаём базу

```
import sqlite3  
  
conn = sqlite3.connect('user_data.db')  
cur = conn.cursor()  
cur.execute("CREATE TABLE IF NOT EXISTS users (user_id INTEGER,  
name TEXT)")  
conn.commit()
```

Записываем имя при команде /setname

```
async def setname_command(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    args = context.args
    if args:
        name = " ".join(args)
        user_id = update.effective_user.id
        cur.execute("INSERT OR REPLACE INTO users (user_id,
name) VALUES (?, ?)", (user_id, name))
        conn.commit()
        await update.message.reply_text(f"Привет, {name}!")
    else:
        await update.message.reply_text("Пожалуйста, укажи своё
имя после /setname")
```

Чтение имени

```
cur.
execute("SELECT name FROM users WHERE user_id=?", (user_id,))
row = cur.fetchone()
if row:
    name = row[0]
    ...
```

4. Получение ID стикеров и их отправка

4.1. Получение file_id присланного стикера

Если пользователь отправляет бот-аккаунту стикер, мы можем «подсмотреть» его `file_id`:

```
from telegram.ext import MessageHandler, filters

async def sticker_handler(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    sticker_obj = update.message.sticker
    if sticker_obj:
        file_id = sticker_obj.file_id
```



```
await update.message.reply_text(f"Sticker ID: {file_id}")
else:
    await update.message.reply_text("Это не похоже на стикер.")
```

Регистрация

```
application.add_handler(MessageHandler(filters.STICKER,
sticker_handler))
```

- Отправьте боту любой стикер, и бот пришлёт вам его `file_id` (уникальный идентификатор).
- Далее этот `file_id` можно использовать в команде `/sendsticker`.

4.2. Отправка стикеров пользователю

Telegram-бот может отправлять пользователю не только текст, но и стикеры. Для этого нужно знать `file_id` стикера.

```
async def send_sticker_command(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    sticker_file_id = "CAACAgIAAxkBAAEC..." # замените на свой
идентификатор
    await update.message.reply_sticker(sticker=sticker_file_id)
```

Регистрация

```
application.add_handler(CommandHandler("sendsticker",
send_sticker_command))
```

После этого при вводе в Telegram-бот команды `/sendsticker` бот отправит заготовленный стикер.

5. Интерактивная работа

5.1. Практическое задание

Обязательная часть:

1. Добавьте команды `/help`, `/info`, `/setname`, `/greet`, `/sendsticker`.
2. Сделайте **Reply-клавиатуру** (хотя бы 1–2 кнопки).
3. Если пользователь отправляет стикер — бот присылает ответ с `file_id`.
4. Сохраните результат.

Дополнительная часть для тех, кто хочет больше практики:

- Попробовать **Inline-кнопку** с **callback_data**, чтобы при нажатии бот отвечал каким-то сообщением или стикером.
- Сохранить имя пользователя в **SQLite**, чтобы после перезапуска бота имя не терялось.

5.2. Подсказки

- Проверяйте логи в терминале (активируйте **logging**, как это было на прошлом занятии), чтобы видеть происходящие события.
- Не забывайте, что в версии **python-telegram-bot 20+** используется асинхронный подход: функции-обработчики команд делайте **async def**, а все методы (как **update.message.reply_text()**) вызывайте через **await**.

6. Часто возникающие вопросы и ответы на них

1. Почему **file_id** иногда меняется или становится недействительным?

Стикерс хранятся на серверах Telegram, и при обновлениях они могут получать новые идентификаторы. Однако для большинства практических случаев один и тот же **file_id** работает достаточно долго.

2. Как обрабатывать нажатия на **Inline-кнопки** внутри бота, а не открывать ссылку?

Нужно использовать **callback_data** и **CallbackQueryHandler**, который будет вызываться при нажатии кнопки.

7. Домашнее задание (опционально)

Задание 1. Добавить несколько стикеров, чтобы бот мог отправлять разные стикеры по разным командам (например, **/sendhappy**, **/sendsad** и т. д.).

Задание 2. Сделать **Inline-кнопку**, при нажатии на которую бот отправляет конкретный стикер (вместо открытия ссылки).

8. Итоги занятия

Сегодня вы научились:

1. Обработать несколько команд. Теперь бот «понимает» не только **/start**, но и **/help**, **/info**, **/setname**, **/greet** и проч.

2. Создавать кнопки:
 - **Reply-клавиатура** (заменяет клавиатуру чата);
 - **Inline-кнопки** (появляются внутри сообщений).
3. Сохранять пользовательские данные:
 - во временном хранилище (словарь Python) — для быстрых тестов;
 - в **SQLite**, чтобы данные не терялись между перезапусками бота.
4. Работать со стикерами:
 - отправлять стикеры, зная их **file_id**;
 - получать **file_id**, если пользователь присылает стикер боту.

Практический результат

- У каждого обучающегося теперь есть более функциональный бот, который взаимодействует с пользователями через команды и кнопки.
- Бот умеет запоминать имена и работать со стикерами, делая общение интереснее.

Поздравляем! Теперь ваш бот распознаёт несколько команд, может сохранять данные пользователя, отправлять и принимать стикеры. А вы можете расширить его функционал встроенными кнопками, чтобы пользователю не нужно было вводить команды. Поэкспериментируйте и решите, какие из новых функций сделают вашего бота более полезным.

На следующем занятии вы научитесь подключать к боту модель GigaChat, которая сможет поддерживать диалог с пользователем в заданном стиле.