

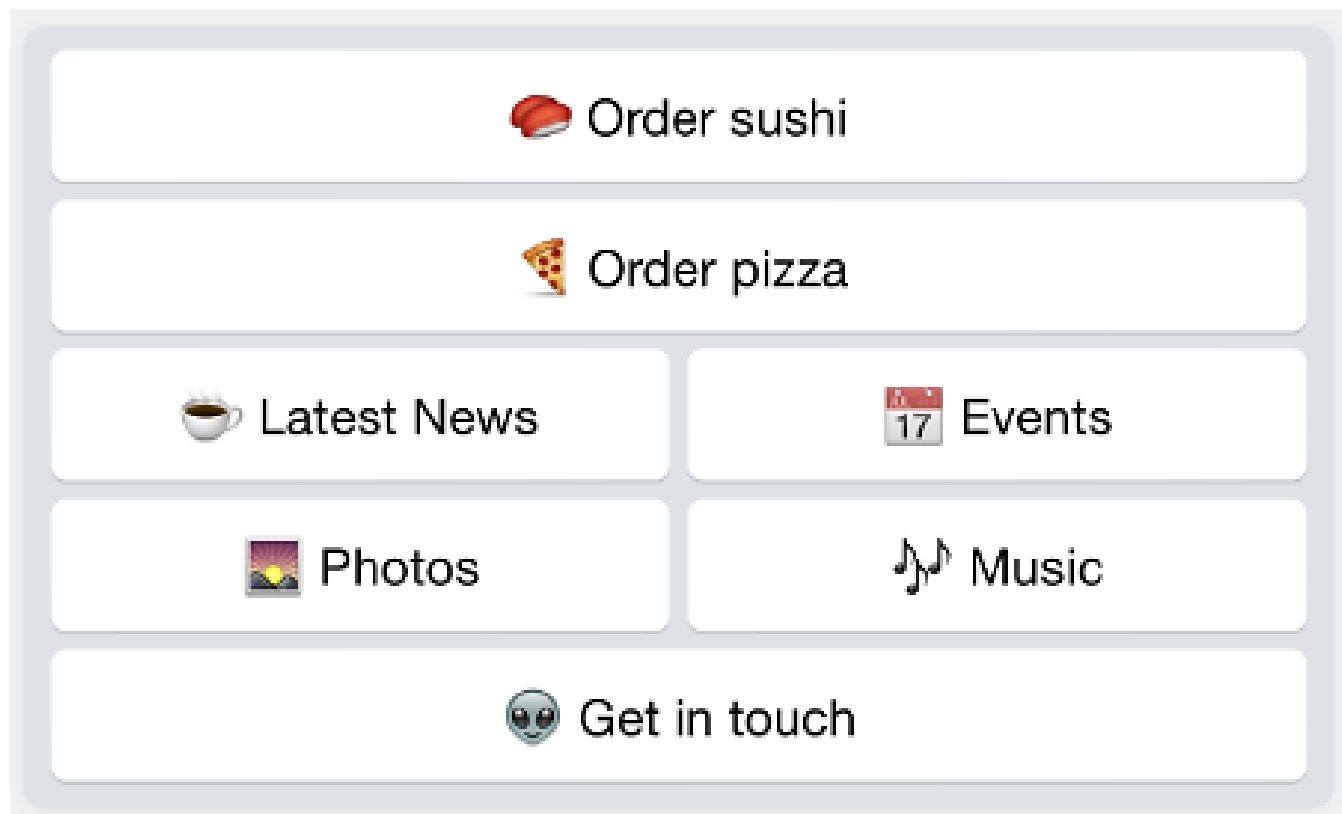
Создание Telegram-бота

Telegram-боты могут быть полноценными веб-приложениями с собственным пользовательским интерфейсом.

Для создания простых ботов можно использовать сервисы-конструкторы. Популярные конструкторы для создания ботов Telegram:

Manybot — подойдет для самых простых сценариев использования;

PuzzleBot — мощный конструктор, позволяющий создать бота практически любой сложности и без единой строчки кода. При этом важно понимать основы программирования.



У Telegram открытое API, поэтому ботов можно разрабатывать практически на любом языке программирования, но удобнее использовать для этого уже готовые библиотеки.



[Гайд от телеграм](#), где приведен список всех существующих библиотек для разработки ботов для всех языков программирования:

Node.js

Swift

C

PHP

Lua

Scala

Python

Go

Perl

Java

C++

Haskell

Ruby

Dart

OCaml

Боты — специальные аккаунты в Telegram, созданные для того, чтобы автоматически обрабатывать и отправлять сообщения. Пользователи могут взаимодействовать с ботами при помощи сообщений, отправляемых через обычные или групповые чаты. Логика бота контролируется при помощи HTTPS запросов к телеграм [API для ботов](#).

Что могут делать боты?

- ✓ **Интеграция с другими сервисами.** Например, бот может отправлять комментарии или управлять «умным домом». Или, например, отправлять вам уведомления при совершении каком-то действия или события (Примеры: [GitHub Bot](#), [Image Bot](#)).
- ✓ **Утилиты и инструменты.** Бот может отображать погоду, переводить тексты или предупреждать о предстоящих событиях по вашему запросу (Например: [бот опросов](#)).
- ✓ **Одно- и многопользовательские игры.** Бот может поиграть с вами в шашки или шахматы, проводить викторины и так далее. (Пример: [Trivia bot](#)).
- ✓ **Социальные сервисы.** Бот может находить вам собеседника, основываясь на ваших общих интересах и увлечениях. (Пример: [HotOrBot](#)).

Как создать бота?

Просто напишите пользователю @BotFather и следуйте его инструкциям. В итоге вы получите свой ключ (токен) авторизации, который будете использовать при создании бота.



Чем бот отличается от обычного аккаунта?

- У роботов нет статусов «онлайн» и «был в сети», вместо этого отображается надпись «бот».
- Для ботов выделено ограниченное место на серверах — все сообщения будут удалены по прошествии определённого срока после обработки.
- Боты не могут сами начать общение с пользователем. Пользователь должен либо добавить робота в группу, либо первым начать с ним диалог.
- Имя пользователя у робота должно заканчиваться на «bot» (например, @controllerbot).
- При добавлении в конференцию, по умолчанию робот не получает всех сообщений (см. режим приватности).

Команды

Команды представляют собой гибкий способ общения с ботом.
Рекомендуется следующий синтаксис:

`/команда [необязательный] [аргумент]`

Команда должна начинаться с символа косой черты «/» и не может быть длиннее 32 символов. Команды могут состоять из букв латинского алфавита, цифр и подчёркивания. Несколько примеров:

`/get_messages_stats`

`/set_timer 10min Alarm!`

`/get_timezone London, UK`

Глобальные команды

Чтобы пользователям было проще работать с ботами, рекомендуется реализовывать поддержку нескольких простых команд.

- ✓ **/start** — начинает общение с пользователем (например, отправляет приветственное сообщение).
- ✓ **/help** — отображает сообщение с помощью по командам. Оно может представлять собой короткое сообщение о боте и список доступных команд.
- ✓ **/settings** — (по возможности) возвращает список возможных настроек и команды для их изменения.

Через @BotFather можно настроить своего бота:

/setname — Изменить имя робота.

/setdescription — Изменить описание робота, представляющее собой короткий текст с описанием бота. Пользователи увидят его в самом начале, под заголовком «Что умеет этот робот?».

/setabouttext — Изменить информацию о боте, ещё более короткий текст, отображающийся в профиле бота. Ещё, если кто-то поделится вашим ботом, то вместе со ссылкой на него отправится этот текст.

/setuserpic — Изменить аватарку бота. Картинки — всегда хорошо.

/setcommands — Изменить список команд бота. Каждая команда состоит из собственно командного слова, начинающегося с символа косой черты («/») и короткого описания. Пользователи увидят список команд при вводе символа «/».

/setjoingroups — Определяет, можно ли добавлять вашего бота в группы.

/setprivacy — Определяет, все ли сообщения видит ваш бот в группах. В выключенном состоянии роботу будут отправляться все сообщения.

/deletebot — Удалить бота и его имя пользователя.



/mybots 7:57 ✓✓

Here it is: first_bot_1133 @Bgty1133_bot.
What do you want to do with the bot?

7:57

API Token

Edit Bot

Bot Settings

Payments

Transfer Ownership

Delete Bot

« Back to Bots List

Для создания Telegram-бота на Python есть несколько популярных библиотек

1. python-telegram-bot (PTB)

Лучший выбор для большинства проектов

Плюсы:

- Официально рекомендована Telegram (telegram-bot-api)
- Асинхронная (asyncio) и синхронная версии
- Хорошая документация и активное сообщество
- Гибкость и мощные возможности
- Поддержка Inline-кнопок, WebApp, Payments и других фич Telegram Bot API

Минусы:

- Сложнее для новичков

2. aiogram

Лучший выбор для асинхронных ботов

Плюсы:

- Полностью асинхронный (asyncio)
- Удобный и современный API
- Хорошая поддержка FSM (Finite State Machine)
- Активное развитие

Минусы:

- Меньше документации по сравнению с PTB
- Нужно знать асинхронность

3. pyTelegramBotAPI (telebot)

Лучший выбор для простых ботов

Плюсы:

- Простота и удобство
- Подходит для новичков
- Есть синхронный и асинхронный режимы

Минусы:

- Менее гибкий, чем PTB и aiogram
- Медленнее обновляется

Пример кода на **python-telegram-bot (PTB)**

```
from telegram import Update
from telegram.ext import Application,
CommandHandler, MessageHandler, filters

async def start(update: Update, context):
    await update.message.reply_text("Привет!")

app = Application.builder().token("TOKEN").build()
app.add_handler(CommandHandler("start", start))
app.run_polling()
```

Пример кода на **telebot**

```
import telebot

bot = telebot.TeleBot("TOKEN")

@bot.message_handler(commands=['start'])
def start(message):
    bot.reply_to(message, "Привет!")

bot.polling()
```

Работа с telebot

```
1.  import telebot # импортируем telebot
2.
3.
4.  # передаём значение переменной с кодом экземпляру бота
5.  token = ('Ваш токен')
6.  bot = telebot.TeleBot(token)
7.
8.
9.  # хендлер и функция для обработки команды /start
10. @bot.message_handler(commands=['start'])
11. def start_message(message):
12.     bot.send_message(message.chat.id, "Привет 🙋")
13.
14.
15. # Бесконечное выполнение кода
16. bot.polling(none_stop=True, interval=0)
```

Здесь все
обработчики
и прочие
функции

1. Обработчики сообщений

```
import telebot
```

```
# Создаём экземпляр бота
```

```
bot = telebot.TeleBot("TOKEN ")
```

```
# Обработчик команды /start
```

```
@bot.message_handler(commands=['start'])
```

```
def send_welcome(message):
```

```
    bot.send_message(message.chat.id, "Привет! Я бот.")
```

```
# Обработчик текстовых сообщений
```

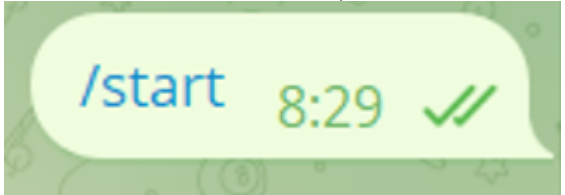
```
@bot.message_handler(func=lambda message: True)
```

```
def echo_all(message):
```

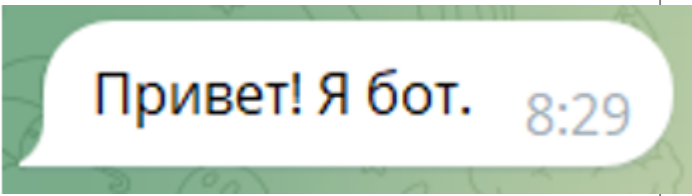
```
    bot.send_message(message.chat.id, message.text)
```

```
# Запускаем бота
```

```
bot.polling()
```



/start 8:29 ✓✓



Привет! Я бот. 8:29



куку 8:40 ✓✓



куку 8:40

```
# Обработчик текстовых сообщений
```

```
@bot.message_handler(func=lambda message: True)
```

Это декоратор, который регистрирует функцию-обработчик для всех входящих сообщений.

`func=lambda message: True` - это параметр декоратора, который определяет условие для фильтрации сообщений:

- `lambda` - это анонимная функция
- `message` - параметр (объект сообщения от пользователя)
- `True` - всегда возвращает `True`

Таким образом, этот обработчик будет:

- Получать все входящие сообщения (так как условие всегда `True`)
- Вызывать декорированную функцию для каждого сообщения

Это аналог "ловушки всего" - обработчик, который перехватывает все сообщения, не пойманные другими, более специфичными обработчиками.

Только на "Привет"

```
@bot.message_handler(func=lambda message: message.text == "Привет")
```

```
def hello_handler(message):
```

```
    bot.send_message(message.chat.id, text="И тебе привет, {0.first_name}! "  
    .format(message.from_user))
```

На все остальные сообщения

```
@bot.message_handler(func=lambda message: True)
```

```
def catch_all(message):
```

```
    bot.send_message(message.chat.id, "Я не понимаю :(")
```

Позволяет достать first
name из аккаунта
пользователя

Объект Message также имеет атрибут **content_type**, который определяет тип Message. content_type может быть следующим:

text, audio, document, animation, game, photo, sticker, video, video_note, voice, location, contact . . .

Подробнее о типах <https://pypi.org/project/pyTelegramBotAPI/#types>

```
# Обработчик для фото
@bot.message_handler(content_types=['photo'])
def handle_photo(message):
    bot.send_message(message.chat.id, "Отличное фото!")
```

```
@bot.message_handler(func=lambda message: True)
```

```
def add_numbers(message):
```

```
    try:
```

```
        numbers = message.text.split() # Разбиваем сообщение на две части
```

```
        if len(numbers) != 2:
```

```
            bot.send_message(message.chat.id, "Пожалуйста, введите два числа через пробел.")
```

```
            return
```

```
        # Преобразуем строки в числа
```

```
        num1 = float(numbers[0])
```

```
        num2 = float(numbers[1])
```

```
        # Складываем и отправляем результат
```

```
        result = num1 + num2
```

```
        bot.reply_to(message, f"Результат сложения: {result}")
```

```
    except ValueError:
```

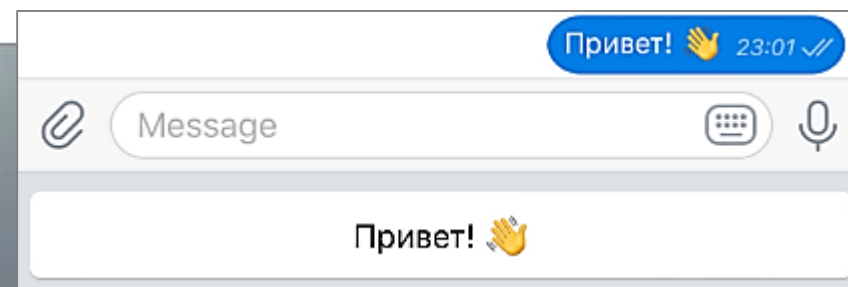
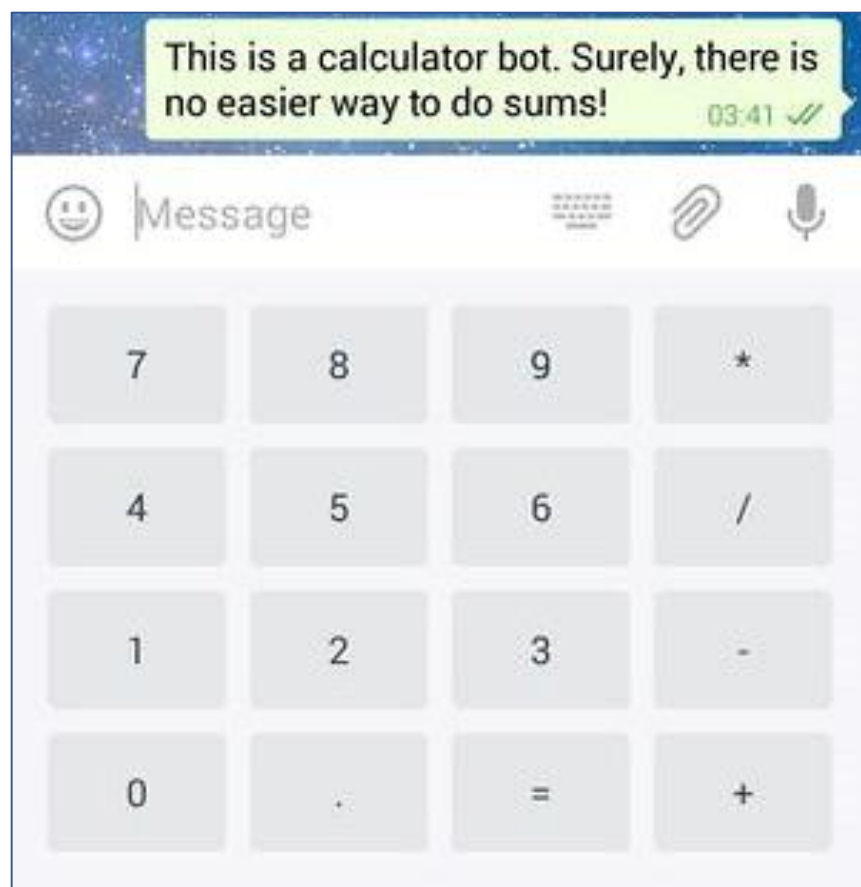
```
        bot.reply_to(message, "Пожалуйста, введите два числа через пробел (например: '5 3').")
```

```
bot.polling() # Запускаем бота
```

Пример обработки введенной строки. Сложение двух чисел.

2. Клавиатуры

Можно создавать кастомную клавиатуру. Нажатие на клавишу отправляет на сервер соответствующую команду. Таким образом можно значительно упростить взаимодействие робота с пользователем. На данный момент для отображения на клавише могут использоваться эмодзи и текст.



2. Клавиатуры

```
import telebot
from telebot import types
bot = telebot.TeleBot("TOKEN")
```

Простая клавиатура Да Нет

```
markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
```

```
itembtn1 = types.KeyboardButton('Да')
```

```
itembtn2 = types.KeyboardButton('Нет')
```

```
markup.add(itembtn1, itembtn2)
```

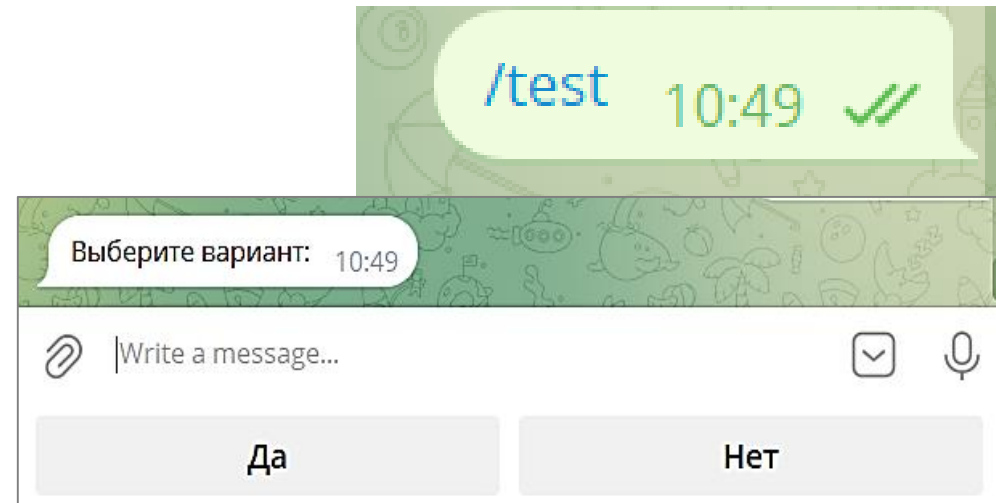
```
@bot.message_handler(commands=['test'])
```

```
def send_welcome(message):
```

```
    bot.send_message(message.chat.id, "Выберите вариант:",
```

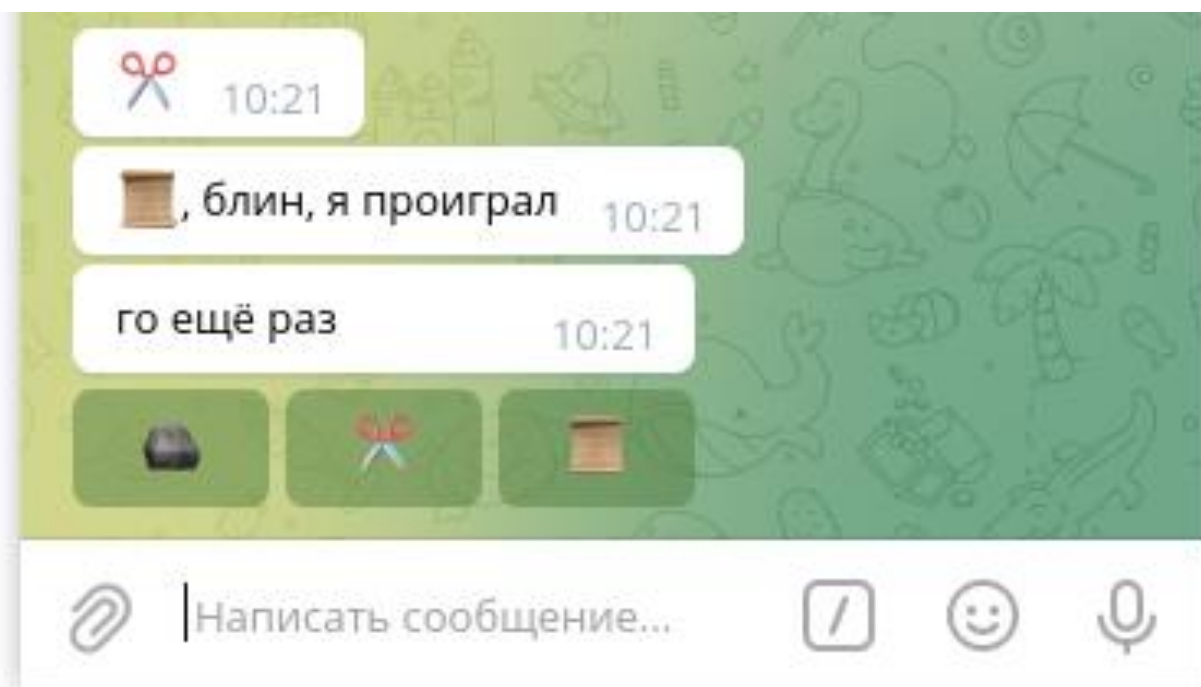
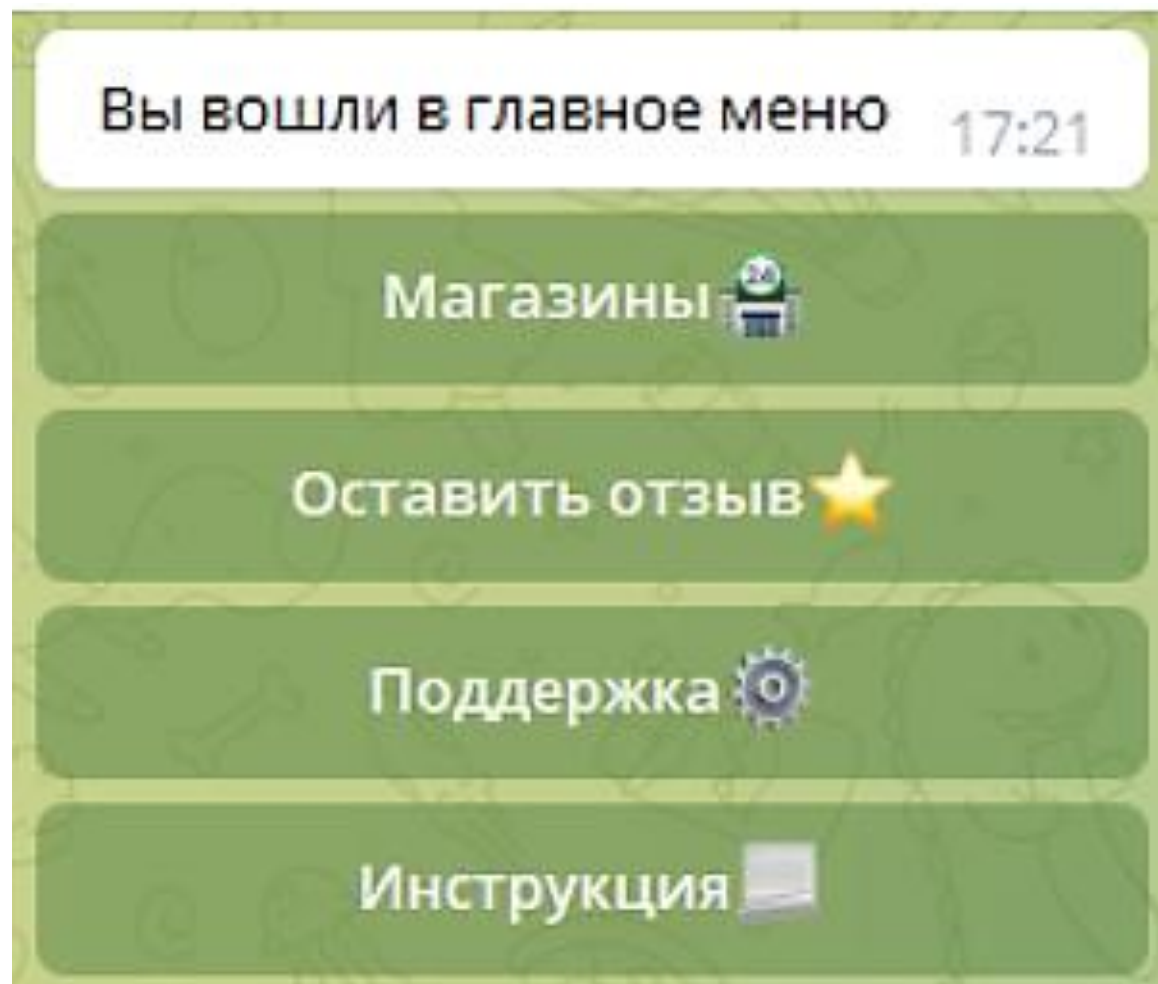
```
    reply_markup=markup)
```

```
bot.polling()
```



```
@bot.message_handler(commands=['start'])
def start_message(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    start_button = types.KeyboardButton("🚀 Старт")
    action_button = types.KeyboardButton("😊 Комплимент")
    markup.add(start_button, action_button)
    bot.send_message(message.chat.id,
                      text="Привет, {0.first_name} 🖐️\nВоспользуйся кнопками".format(message.from_user),
                      reply_markup=markup)
# хендлер для обработки нажатий кнопок
@bot.message_handler(content_types=['text'])
def buttons(message):
    if (message.text == "🚀 Старт"):
        bot.send_message(message.chat.id, text="Я могу поддержать тебя и поднять настроение.")
    elif (message.text == "😊 Комплимент"):
        bot.send_message(message.chat.id, text="Я не встречал людей лучше тебя!")
```

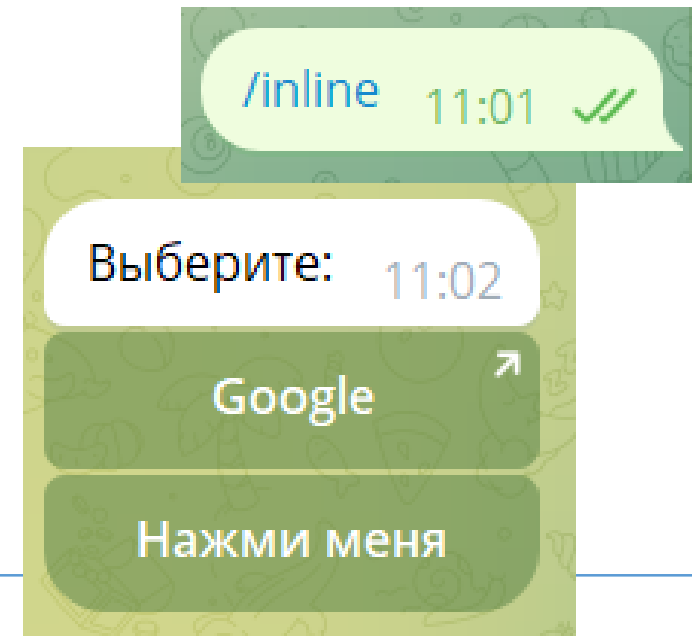
3. Inline-кнопки



3. Inline-кнопки

```
@bot.message_handler(commands=['inline'])
def send_inline(message):
    markup = types.InlineKeyboardMarkup()
    markup.add(types.InlineKeyboardButton(text='Google', url='https://google.com'))
    markup.add(types.InlineKeyboardButton(text='Нажми меня', callback_data='press'))
    bot.send_message(message.chat.id, "Выберите:", reply_markup=markup)

# Обработчик callback-запросов от inline-кнопок
@bot.callback_query_handler(func=lambda call: True)
def callback_query(call):
    if call.data == "press":
        bot.answer_callback_query(call.id, "Вы нажали кнопку!")
```



Последовательный вызов функций

```
1
@bot.message_handler(commands=['start'])
def start_registration(message):
    msg = bot.send_message(message.chat.id, text: "Введите ваше имя:")
    bot.register_next_step_handler(msg, process_name)
2
def process_name(message): 1 usage
    user_data[message.chat.id] = {'name': message.text}
    msg = bot.send_message(message.chat.id, text: "Введите ваш возраст:")
    bot.register_next_step_handler(msg, process_age)
3
def process_age(message): 2 usages
    if not message.text.isdigit():
        msg = bot.send_message(message.chat.id, text: "Возраст должен быть ч
        bot.register_next_step_handler(msg, process_age)
    return
```


4. Где хранить данные

1. Временные данные (в оперативной памяти)

Используйте: Обычные структуры Python (**dict**, **list** и т. д.).

```
user_data = {}

@bot.message_handler(commands=['start'])
def start_registration(message):
    msg = bot.send_message(message.chat.id, text: "Введите ваше имя:")
    user_data[message.chat.id] = {'name': message.text}
```

Плюсы:

- Очень быстро.
- Простота реализации.

Минусы:

- Данные теряются при перезапуске бота.
- Не подходит для больших объемов данных.

4. Где хранить данные

2. Файлы (JSON, CSV)

Подходит для: Небольших ботов, где не требуется высокая скорость работы.

Плюсы:

- Человекочитаемый формат.
- Простота использования.

Минусы:

- Медленный при больших данных.
- Нет структурированного поиска.

```
import json

# Сохранение
data = {"user_id": 123, "name": "Alice"}
with open("data.json", "w") as f:
    json.dump(data, f)

# Загрузка
with open("data.json", "r") as f:
    loaded_data = json.load(f)
```

4. Где хранить данные

3. SQLite (встроенная БД)

Плюсы:

- Надежнее файлов.
- Поддержка SQL-запросов.

Минусы:

- Требуется знание SQL.

```
import sqlite3

conn = sqlite3.connect("bot_data.db")
cursor = conn.cursor()
cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER, name TEXT)")
cursor.execute("INSERT INTO users VALUES (?, ?)", (123, "Alice"))
conn.commit()
conn.close()
```

4. Где хранить данные

4. Внешние базы данных (PostgreSQL, MySQL, MongoDB)

Подходит для средних и крупных ботов с большим числом пользователей.

PostgreSQL / MySQL

Плюсы:

- Высокая производительность.
- Поддержка транзакций.

Минусы:

- Требуется сервер БД.

4. Где хранить данные

5. Кэширование (Redis)

Подходит для: Быстрого доступа к временным данным (например, состояния FSM).

Плюсы:

Очень высокая скорость.

Поддержка TTL (автоудаление данных).

Минусы:

Данные в памяти (нужно дублировать в постоянное хранилище).

4. Где хранить данные

6. Облачные хранилища (Firebase, Supabase)

Подходит для: Ботов, развернутых в облаке.

Плюсы:

Облачное решение, не требует своего сервера.

Масштабируемость.

Минусы:

Зависимость от интернета.

Рекомендации:

Для маленьких ботов → **SQLite** или **JSON**.

Для средних ботов → **PostgreSQL** или **MongoDB**.

Для быстрого кэширования → **Redis**.

Для облачных решений → **Firebase/Firestore**.

Итого

ОС	Ubuntu 22.04
Регион	Санкт-Петербург
CPU	1 x 3.3 ГГц
RAM	1 ГБ
NVMe	15 ГБ
Конфигурация	188 ₽/мес
Количество серверов	1

Деплой

Для того чтобы ваш бот был всегда онлайн, не зависел от вашего устройства, интернета и т.д. необходимо арендовать виртуальный сервер.

Есть много компаний которые позволяет вам арендовать сервер, в статье приведен пример регистрации на одном из самых доступных ХОСТИНГОВ [Timeweb](https://timeweb.ru) (хостинг-компания - компания занимающаяся продажей/арендой серверов, доменов, сетей и т.д.).

<https://habr.com/ru/articles/778182/>