

S4: Интерфейс бота: меню-клавиатуры, inline-кнопки, сценарии ввода

Цель: создать интерактивный пользовательский интерфейс для нашего бота с клавиатурами, кнопками и сценариями ввода



План семинара

01

Разогрев и связка с предыдущими занятиями

Повторение основ и подготовка к новому материалу

02

Парсинг чисел из текста

Обработка пользовательского ввода с разделителями

03

Reply-клавиатуры и сценарии

Создание меню и обработка взаимодействий

04

Inline-кнопки и логирование

Интерактивные элементы и диагностика

05

Мини-квиз и домашнее задание

Закрепление материала и задачи для самостоятельной работы

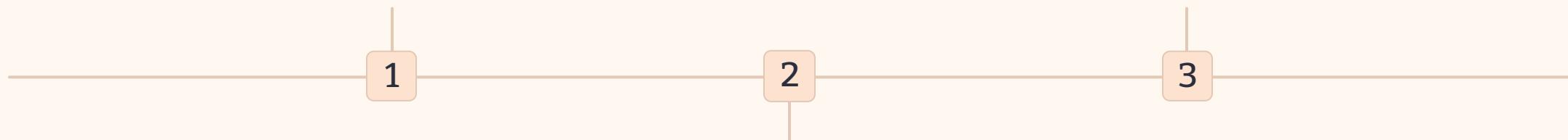
Связка с предыдущими занятиями

S1: Основы

Настройка окружения, создание Hello Bot, получение токена,
первый запуск

S4: Интерфейс

Клавиатуры, кнопки, сценарии ввода, улучшение
пользовательского опыта



S2-S3: Команды

Основы Python, реализация /about, /sum, разбор аргументов
командной строки

Сегодня мы превратим простого бота в интерактивное приложение с удобным интерфейсом. Будем использовать всё, что изучили ранее, и добавим новые возможности для взаимодействия с пользователями.

Как работает наш бот: архитектурная схема

Пользователь пишет

Серверы Telegram

Long Polling

Обработчики команд

Long Polling

Идеально подходит для разработки и тестирования на локальной машине. Бот периодически опрашивает серверы Telegram на наличие новых сообщений.

Библиотека TeleBot

Используем `pyTelegramBotAPI` (`TeleBot`) в синхронном режиме — простой и понятный подход для начинающих.

Ограничения нашего курса



Python 3.11
Фиксированная версия для
единообразия на всех машинах



pyTelegramBotAPI
Библиотека TeleBot, синхронный режим
работы



Long Polling
Без webhooks, Docker, CI/CD — только
локальная разработка



Токен в .env
Безопасное хранение, никогда не
коммитим в Git



SQLite позже
База данных не входит в тему S4, будет в
следующих занятиях



Парсер чисел: требования

Нам нужно научиться извлекать числа из пользовательского ввода, поддерживая разные форматы и игнорируя лишние символы. Это основа для команд вроде `/sum`.

Поддержка разделителей

Пробелы и запятые: "2, 3, 5" или "10 20 30"

Отрицательные числа

Корректная обработка: "2, -3, 5" → [2, -3, 5]

Игнорирование команд

`"/sum 2 3"` → извлекаем только [2, 3]

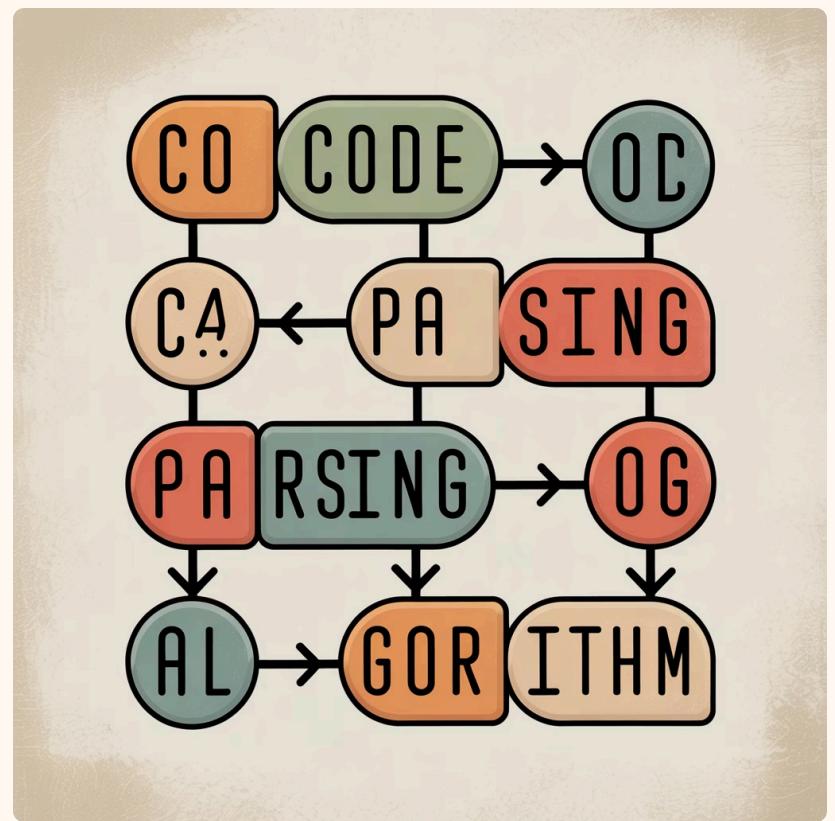
Фильтрация мусора

"2, 3, x, -1" → берём только валидные числа

Парсер чисел: реализация

💡 Алгоритм работы парсера (c list comprehensions)

```
def parse_ints_from_text(text: str) -> list[int]:  
    # Заменяем запятые на пробелы  
    text = text.replace(",", " ")  
  
    # Разбиваем на токены, исключаем команды  
    tokens = [t for t in text.split()  
              if not t.startswith("/")]  
  
    # Оставляем только валидные числа  
    return [int(t) for t in tokens  
           if t.strip().lstrip("-").isdigit()]
```



💡 Алгоритм работы парсера (версия для начинающих)

```
def parse_ints_from_text_beginner(text: str) -> list[int]:  
    # Заменяем запятые на пробелы  
    text = text.replace(",", " ")  
  
    # Разбиваем на токены, исключаем команды  
    all_tokens = text.split()  
    tokens = []  
    for t in all_tokens:  
        if not t.startswith("/":)  
            tokens.append(t)  
  
    # Оставляем только валидные числа  
    result = []  
    for t in tokens:  
        cleaned_token = t.strip().lstrip("-")  
        if cleaned_token.isdigit():  
            result.append(int(t))  
    return result
```

Ключевые моменты:

- `replace(", ", " ")` унифицирует разделители
- `lstrip("-")` обрабатывает отрицательные числа
- `isdigit()` проверяет валидность

Обе функции выполняют одну и ту же задачу, но вторая версия с обычными циклами `for` и `if` может быть более понятной для новичков в программировании.

🧪 Практика: тестируем парсер через /sum

1

Запустите бота

python main.py в терминале

2

Протестируйте команду

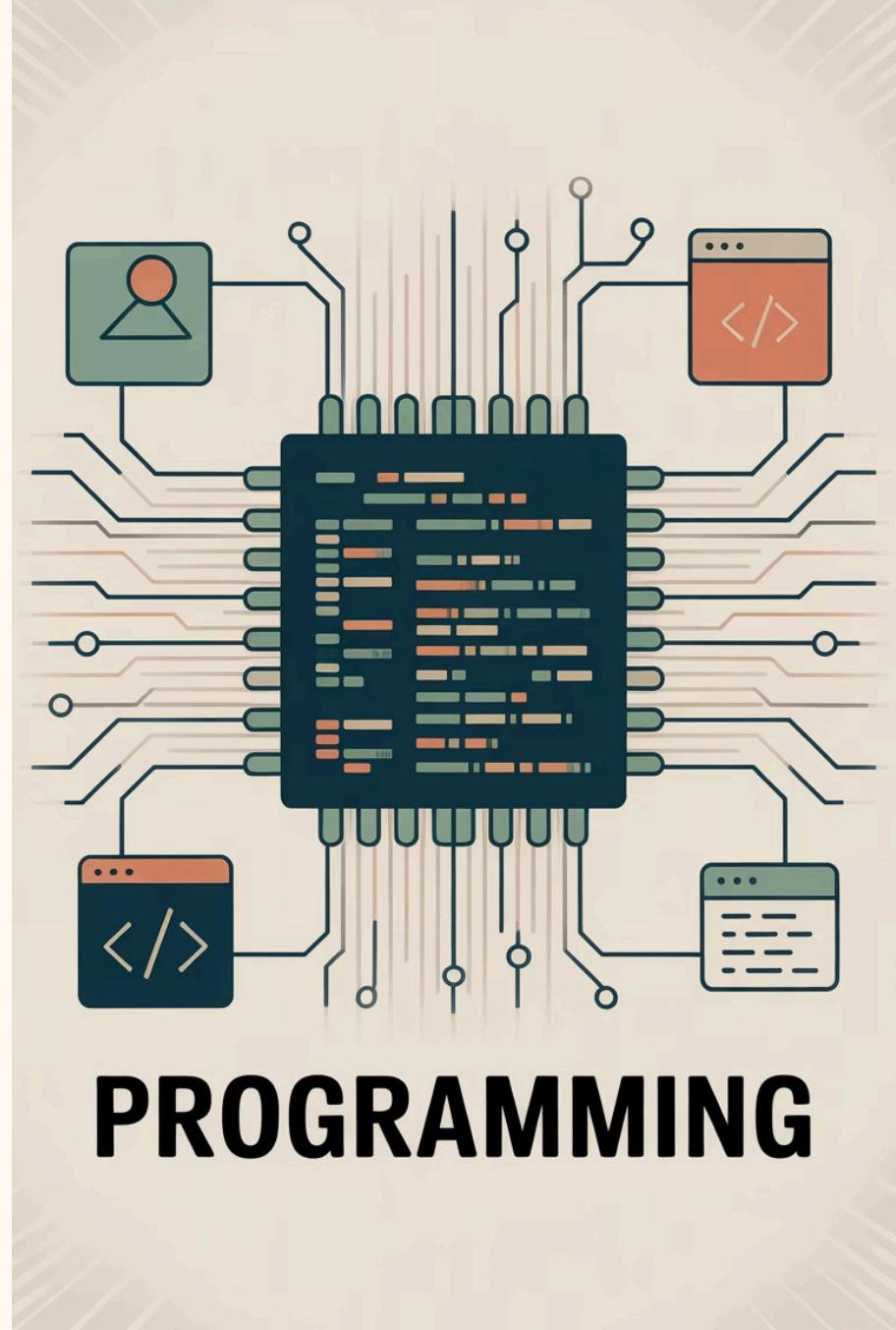
/sum 2, 3, -5 в Telegram

3

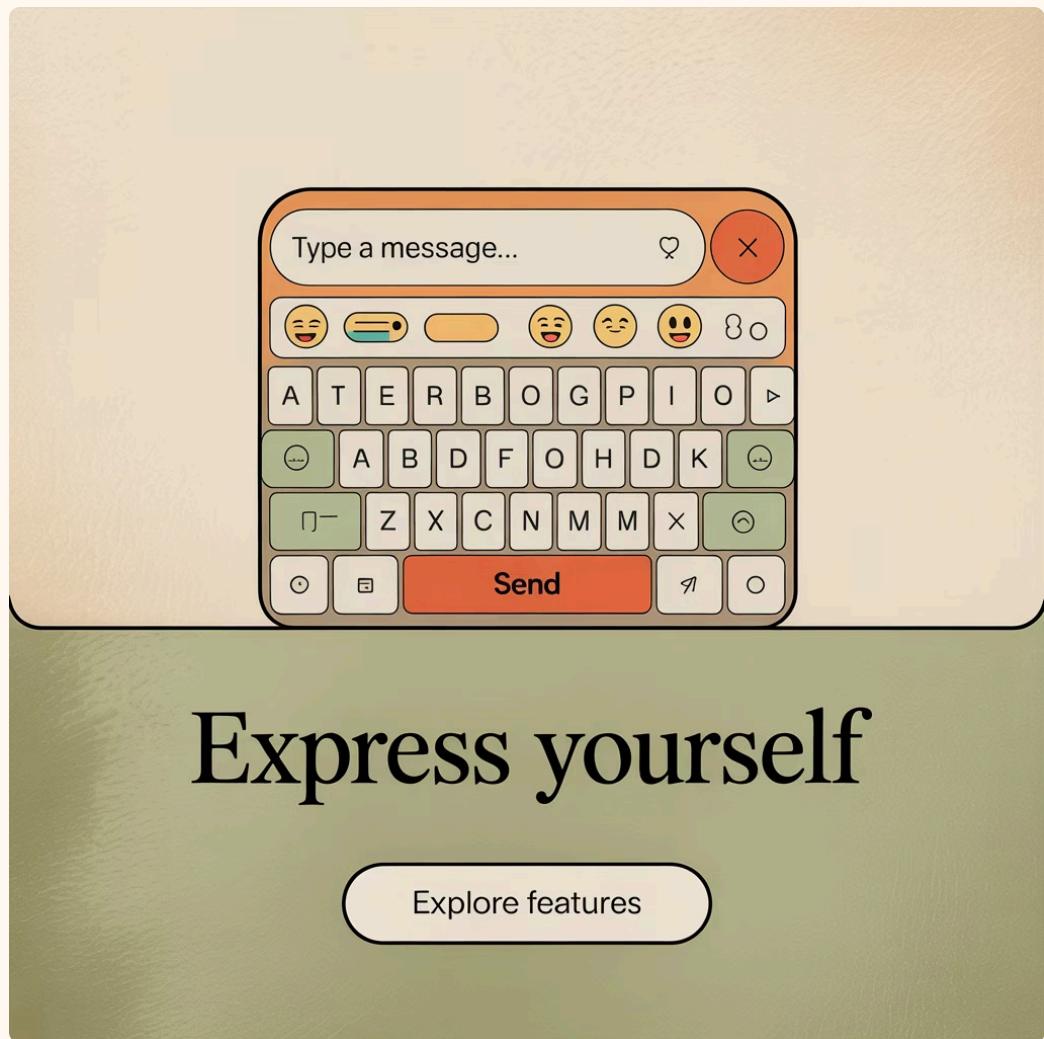
Проверьте результат

Должно вывести "Сумма: 0"

- ☐ ⚠️ Если не работает — проверьте: загружен ли .env с load_dotenv(), правильный ли токен, нет ли проблем с кодировкой файлов



Что такое Reply-клавиатуры



Express yourself

[Explore features](#)

Основные характеристики

Reply-клавиатура появляется над полем ввода сообщения и остаётся видимой до тех пор, пока её не скроют или не заменят.

Преимущества:

- Всегда под рукой у пользователя
- Идеально для основного меню
- Интуитивно понятна
- Экономит время пользователя

В отличие от inline-кнопок, которые привязаны к конкретному сообщению, reply-клавиатура — это постоянное меню для быстрого доступа к основным функциям бота.

Создание меню: код реализации

```
from telebot import types

def make_main_kb() -> types.ReplyKeyboardMarkup:
    # Создаём клавиатуру с автоподгонкой размера
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)

    # Добавляем кнопки по рядам
    kb.row("О боте", "Сумма")
    kb.row("/help")

    return kb
```



Включаем меню в /start и /help

```
@bot.message_handler(commands=['start','help'])  
def start_help(m):  
    welcome_text = """Привет!
```

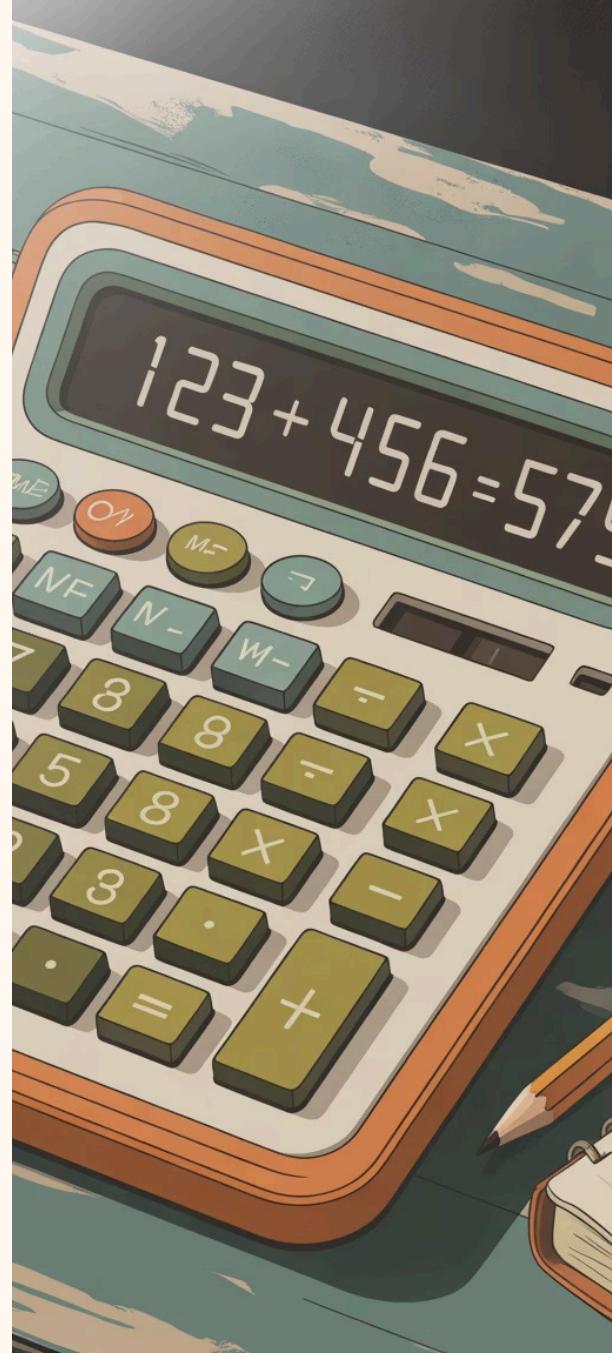
Обработка нажатий кнопок меню

```
@bot.message_handler(func=lambda m: m.text == "О боте")
def kb_about(m):
    bot.reply_to(m, "Я учебный бот: /start, /help, /about, /sum, /echo")

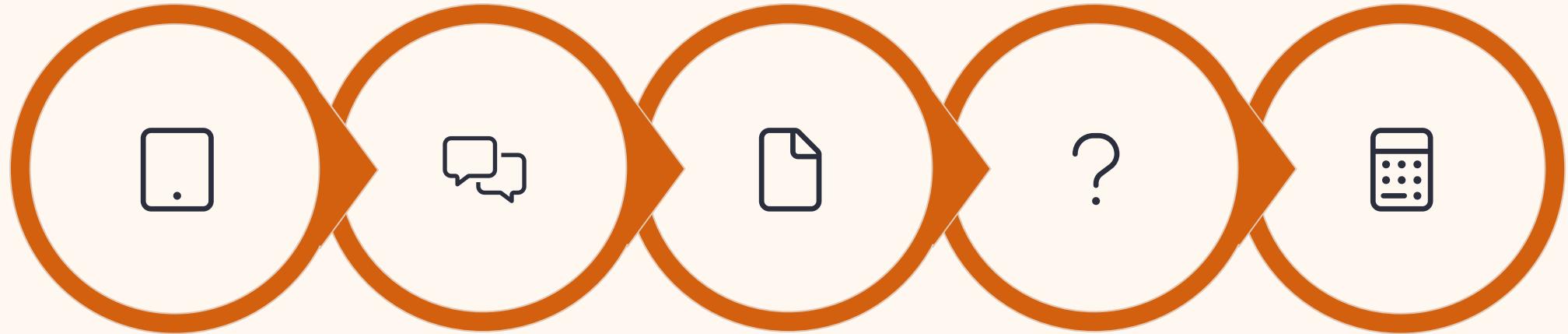
@bot.message_handler(func=lambda m: m.text == "Сумма")
def kb_sum(m):
    bot.send_message(m.chat.id, "Введи числа через пробел или запятую:")
    bot.register_next_step_handler(m, on_sum_numbers)
```

Сценарий «Сумма»: обработка ввода

```
def on_sum_numbers(m):
    nums = parse_ints_from_text(m.text)
    if not nums:
        bot.reply_to(m, "Не вижу чисел. Пример: 2 3 10 или 2, 3, -5")
    else:
        bot.reply_to(m, f"Сумма: {sum(nums)}")
```



Блок-схема сценария «Сумма»



Нажать
«Сумма»

Запрос чисел

Парсинг
ввода

Есть числа?

Вернуть
сумму

- ☐💡 UX-совет: в ветке «нет чисел» можно снова вызвать register_next_step_handler для повторного запроса

Скрытие клавиатуры

```
@bot.message_handler(commands=['hide'])
def hide_kb(m):
    rm = types.ReplyKeyboardRemove()
    bot.send_message(m.chat.id, "Спрятал клавиатуру.", reply_markup=rm)
```

Reply vs Inline: сравнение подходов

Критерий	Reply-клавиатуры	Inline-кнопки
Где живут	Панель ввода (постоянно)	Внутри конкретного сообщения
Обработка	По тексту сообщения	Через callback_query
Типичные кейсы	Основное меню, навигация	Подтверждения, фильтры, страницы
UX	Всегда под рукой	Контекстные действия
Визуал	Стандартные кнопки ОС	Стильные кнопки в сообщении

Выбор между reply и inline зависит от задач: reply для постоянного доступа, inline для контекстных действий.



Inline-кнопки: создание

```
@bot.message_handler(commands=['confirm'])  
def confirm_cmd(m):  
    kb = types.InlineKeyboardMarkup()  
    kb.add(  
        types.InlineKeyboardButton("Да", callback_data="confirm:yes"),  
        types.InlineKeyboardButton("Нет", callback_data="confirm:no"),  
    )  
    bot.send_message(m.chat.id, "Подтвердить действие?", reply_markup=kb)
```

Обработка callback от inline-кнопок

```
@bot.callback_query_handler(func=lambda c: c.data.startswith("confirm:"))
def on_confirm(c):
    # Извлекаем выбор пользователя
    choice = c.data.split(":", 1)[1] # "yes" или "no"

    # Показываем "тик" на нажатой кнопке
    bot.answer_callback_query(c.id, "Принято")

    # Убираем inline-кнопки
    bot.edit_message_reply_markup(c.message.chat.id, c.message.message_id, reply_markup=None)

    # Отправляем результат
    bot.send_message(c.message.chat.id, "Готово!" if choice == "yes" else "Отменено.")
```

Ключевые функции для inline-кнопок

1

`answer_callback_query()`

Показывает "тик" или всплывающее
уведомление при нажатии кнопки.
Обязательно вызывать для хорошего UX.

2

`edit_message_reply_markup()`

Изменяет или удаляет inline-клавиатуру у
существующего сообщения. `None`
убирает все кнопки.

3

`callback_data` разбор

Используем префиксы вида
`"action:param"` для группировки и
фильтрации разных типов кнопок.



🧪 Практика: модифицируем confirm

1 Поменяйте текст вопроса

Вместо "Подтвердить действие?" напишите "Сохранить изменения?"

2 Смените префикс callback_data

Замените "confirm:" на "save:" (и в кнопках, и в хэндлере)

3 Добавьте третью кнопку

"🕒 Позже" с callback_data="save:later"

4 Протестируйте изменения

Запустите бота и проверьте команду /confirm

Зачем нужно логирование?



Преимущества логирования

Логи помогают понимать, что происходит с ботом в реальном времени, особенно когда пользователей много.

- **Отладка:** видим, какие команды вызывались
- **Мониторинг:** отслеживаем активность пользователей
- **Анализ:** какие числа распарсились в /sum
- **Безопасность:** обнаружение подозрительной активности

Настройка логирования: код

```
import logging

# Базовая настройка логирования
logging.basicConfig(
    level=logging.INFO,
    format"%(asctime)s - %(levelname)s - %(message)s"
)

@bot.message_handler(commands=['sum'])
def cmd_sum(m):
    # Логируем входящую команду
    logging.info(f"/sum от {m.from_user.first_name} {m.from_user.id}: {m.text}")

    nums = parse_ints_from_text(m.text)

    # Логируем результат парсинга
    logging.info(f"распознаны числа: {nums}")

    # Отправляем ответ
    bot.reply_to(m, f"Сумма: {sum(nums)}" if nums else "Пример: /sum 2 3 10")
```



Пример вывода логов

```
2024-01-15 14:30:25,123 - INFO - /sum от Иван 12345: /sum 2, 3, -5
2024-01-15 14:30:25,124 - INFO - распознаны числа: [2, 3, -5]
2024-01-15 14:30:25,125 - INFO - результат: 0
2024-01-15 14:30:32,456 - INFO - /sum от Мария 67890: /sum abc def
2024-01-15 14:30:32,457 - INFO - распознаны числа: []
2024-01-15 14:30:45,789 - INFO - /confirm от Алексей 11111
```

Логи показывают временные метки, имена пользователей, ID, введённые команды и результаты обработки. Это invaluable для debugging и мониторинга.



🎯 Мини-квиз: проверим понимание

“ Вопрос 1

Чем Reply-клавиатуры отличаются от Inline-кнопок и когда что выбирать?

“

Вопрос 2

Как спрятать Reply-клавиатуру и зачем это может понадобиться?

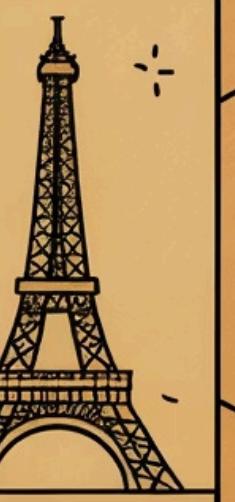
”

“ Вопрос 3

Почему в парсере чисел есть replace(", ", " ") и lstrip("-")?

”

Обсудим ответы устно. Это поможет закрепить ключевые концепции перед переходом к домашнему заданию.



What is the capital of
FRANCE?



When time allows



What is the capital of
FRANCE

🎯 Мини-квиз: проверим понимание

“ Вопрос 1

Чем Reply-клавиатуры отличаются от Inline-кнопок и когда что выбирать?

Ответ: Reply-клавиатуры живут в панели ввода постоянно и обрабатываются как обычные сообщения. Inline-кнопки привязаны к конкретному сообщению и обрабатываются через callback_query. Reply используем для основного меню и навигации, inline — для подтверждений и контекстных действий.

“ Вопрос 2

Как спрятать Reply-клавиатуру и зачем это может понадобиться?

Ответ: Используем types.ReplyKeyboardRemove() в параметре reply_markup. Нужно для очистки интерфейса, когда клавиатура больше не актуальна, или для переключения между разными режимами работы бота.

“ Вопрос 3

Почему в парсере чисел есть replace(", ", " ") и lstrip("-")?

Ответ: replace(", ", " ") унифицирует разделители — превращает запятые в пробелы для единообразной обработки. lstrip("-") убирает минус перед проверкой isdigit(), чтобы корректно распознавать отрицательные числа.

Обсудим ответы устно. Это поможет закрепить ключевые концепции перед переходом к домашнему заданию.



Домашнее задание №2



Команды

Реализовать команду /max (не делали на семинаре). Те же правила парсинга, ответ в виде — "Максимум: X"



Reply-меню

Добавить кнопки: /about, /sum, /hide
/show



Inline-кнопки

Команда /confirm с разными вариантами ответов на выбор



Документация

Обновить README.md: "Как запустить",
"Список команд"



Git

Омысленные коммиты и push в GitHub



Финальный проект

Подумать и определить тему финального проекта

Типичные ошибки и быстрые решения

1

Клавиатура не появляется

✓ Убедитесь, что передаёте reply_markup=make_main_kb() в send_message

2

Числа не парсятся

✓ Проверьте replace(", ", " ") и lstrip("-"), игнорирование токенов с "/"

3

Callback не обрабатывается

✓ Проверьте func-фильтр в хэндлере и соответствие callback_data

4

Токен не подхватывается

✓ Вызовите load_dotenv() до создания bot = TeleBot()

5

Старые сообщения в поллинге

✓ Используйте infinity_polling(skip_pending=True)

Приложение А: Каркас main.py

```
# Основные импорты
import os
import logging
from telebot import TeleBot, types
from dotenv import load_dotenv

# Загрузка переменных окружения
load_dotenv()
bot = TeleBot(os.getenv('TELEGRAM_BOT_TOKEN'))

# Настройка логирования
logging.basicConfig(level=logging.INFO,
                    format"%(asctime)s - %(levelname)s - %(message)s")

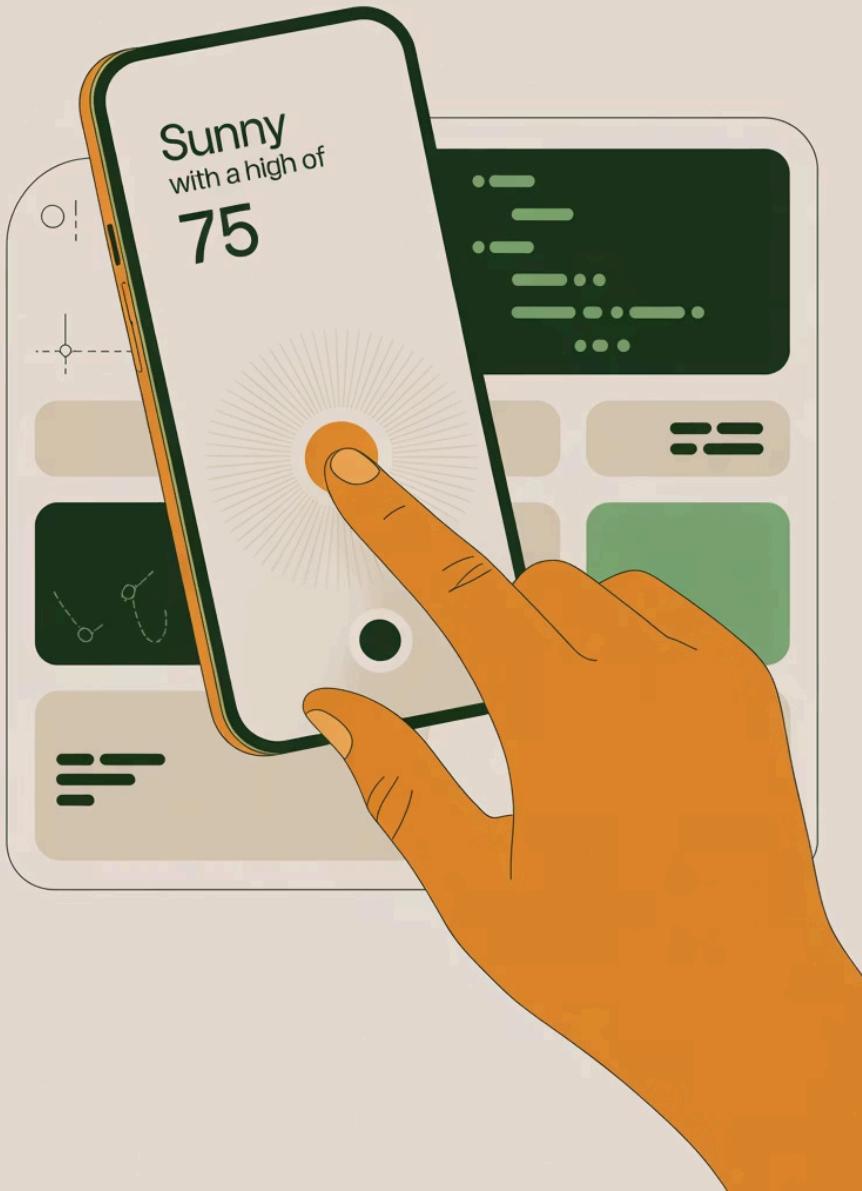
# Функции
def parse_ints_from_text(text: str) -> list[int]:
    # ... (см. слайд 7)

def make_main_kb() -> types.ReplyKeyboardMarkup:
    # ... (см. слайд 10)

# Обработчики команд
@bot.message_handler(commands=['start','help'])
def start_help(m):
    # ... (см. слайд 11)

# ... остальные хэндлеры ...

if __name__ == '__main__':
    bot.infinity_polling(skip_pending=True)
```



Что делаем: Бонус «Погода»

01

Добавляем Reply-кнопку

Создадим кнопку «Погода (Москва)» в Reply-клавиатуре.

02

Запрос к Open-Meteo API

По нажатию кнопки делаем один запрос к Open-Meteo Weather Forecast API, используя фиксированные координаты Москвы.

03

Формируем ответ

Бот отправляет ответ: «Москва: сейчас N °C, описание». Без использования геокодера или ввода города.



Open-Meteo не требует API-ключа для некоммерческого использования; координаты и список нужных полей передаются параметрами запроса.

Эндпоинт и параметры Open-Meteo API

Для получения погодных данных будем использовать следующий эндпоинт:

<https://api.open-meteo.com/v1/forecast>

Обязательные параметры

- `latitude`, `longitude` — координаты (широта и долгота) для определения местоположения.

Полезные параметры

- `current` — список переменных текущих условий, которые нужно получить (например, `temperature_2m` для температуры, `weather_code` для типа погоды).
- `timezone` — устанавливает часовой пояс для возвращаемого времени (например, `Europe/Moscow`).
- `temperature_unit=celsius` — единица измерения температуры (по умолчанию уже $^{\circ}\text{C}$).

Важно: параметр `apikey` нужен только для коммерческих тарифов и не требуется для бесплатного доступа.

- Документация Open-Meteo прямо указывает, что параметр `current` принимает список переменных, описывающих «текущие условия» погоды.

Код: функция получения погоды для Москвы

Эта функция реализует логику запроса данных о погоде для Москвы через Open-Meteo API. Она обрабатывает ответ, извлекает текущую температуру и код погоды, а затем форматирует их в читаемое сообщение.

```
# weather_moscow.py (или прямо в main.py)
import requests

# WMO-коды → краткое описание (минимум нужных)
WMO_DESC = {
    0: "ясно", 1: "в осн. ясно", 2: "переменная облачность", 3: "пасмурно",
    45: "туман", 48: "изморозь", 51: "морось", 53: "морось", 55: "сильная морось",
    61: "дождь", 63: "дождь", 65: "сильный дождь", 71: "снег",
    80: "ливни", 95: "гроза"
}

def fetch_weather_moscow_open_meteo() -> str:
    url = "https://api.open-meteo.com/v1/forecast"
    params = {
        "latitude": 55.7558, "longitude": 37.6173,      # Москва
        "current": "temperature_2m,weather_code",
        "timezone": "Europe/Moscow"
    }
    try:
        r = requests.get(url, params=params, timeout=5)
        r.raise_for_status()
        cur = r.json()["current"]
        t = round(cur["temperature_2m"])
        code = int(cur.get("weather_code", 0))
        return f"Москва: сейчас {t}°C, {WMO_DESC.get(code, 'по данным модели')}"
    except requests.exceptions.RequestException:
        return "Не удалось получить погоду (сеть)."
    except (KeyError, TypeError, ValueError):
        return "Ответ погоды в неожиданном формате."
```

Словарь `WMO_DESC` позволяет преобразовать числовые коды погоды в понятные описания, а блоки `try-except` обеспечивают отказоустойчивость при проблемах с сетевым запросом или неверным форматом данных.

Код: Интеграция кнопки «Погода»

Теперь, когда функция получения погоды готова, интегрируем её в бота. Мы добавим новую кнопку «Погода (Москва)» в главное Reply-меню и настроим её обработку. При нажатии бот будет вызывать нашу функцию и отправлять результат пользователю.

Изменение клавиатуры

```
from telebot import types

def make_main_kb() -> types.ReplyKeyboardMarkup:
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row("О боте", "Сумма")
    kb.row("Погода (Москва)", "/help") # ← новая кнопка
    return kb
```

Обработчик кнопки

```
@bot.message_handler(func=lambda m: m.text == "Погода (Москва)")
def kb_weather_moscow(m):
    bot.send_message(m.chat.id, fetch_weather_moscow_open_meteo())
```

Лимиты и аккуратность API Open-Meteo



Лимиты использования

Для некоммерческого использования Free API устанавливает ограничения: до 10 000 запросов в день, 5 000 в час и 600 в минуту. Будьте внимательны к этим пределам, чтобы обеспечить стабильную работу бота.



Оптимальная частота

Обновляйте данные о погоде экономно. Для одной локации достаточно одного запроса раз в 10–60 минут, чтобы получить актуальную информацию и не превышать установленные лимиты.



Условия лицензии

API-ключ не требуется, но данные распространяются по лицензии CC-BY 4.0. Обязательно ознакомьтесь с условиями использования Open-Meteo и соблюдайте их.

Что можно улучшить

После основной реализации функции получения погоды, можно рассмотреть следующие опциональные улучшения:



Эмодзи по температуре

Добавьте визуальные эмодзи в зависимости от диапазона температуры, чтобы сделать сообщение о погоде более интуитивным и привлекательным для пользователя.



Небольшой кеш

Внедрите простой механизм кэширования (например, словарь с временными метками) на 5-10 минут, чтобы сократить количество запросов к API и не превышать лимиты.



Расширенные данные

Расширьте список запрашиваемых параметров API (`current`) для получения более полной информации, такой как скорость ветра (`wind_speed_10m`) или относительная влажность (`relative_humidity_2m`).



Спасибо за внимание! 🎉

Дедлайны

Домашнее задание №2 нужно сдать до следующего семинара. Не откладывайте — практика закрепляет теорию!

Контакты

Вопросы по домашке и материалу задавайте в общем чате курса или в личных сообщениях в Telegram.

Следующая тема: работа с базами данных SQLite, сохранение состояния пользователей, более сложные сценарии взаимодействия.
