ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ МОЛДОВЫ
ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ
ДЕПАРТАМЕНТ ИНФОРМАТИКИ

Отчет
по дисциплине „ПРОГРАММИРОВАНИЕ В PYTHON”

Руководитель:Плешка Наталья, лектор

Выполнил: студент группы I2302
Богданов Юрий

Кишинев, 2024

Краткая постановка задачи:

Создать Telegram-бота для управления личными финансами, который предоставляет функции конвертации валют и планирования бюджета.

1. *Конвертация валют:*
    - Пользователь может ввести сумму денег.
    - Бот предлагает выбрать валютные пары для конвертации (например, USD/GBP, EUR/BGN, EUR/USD).
    - Пользователь может выбрать из предложенных валютных пар или ввести свою.
    - Бот производит конвертацию суммы и выводит результат.
    - Пользователь может продолжить конвертацию с новой суммой или перейти к планировщику бюджета.
2. *Планировщик бюджета:*
    - Пользователь вводит свой доход.
    - Пользователь вводит свои расходы (с указанием описания и суммы).
    - Бот рассчитывает общий доход, общий расход и оставшийся баланс.
    - Бот дает совет пользователю в зависимости от баланса (положительный или отрицательный).
    - Пользователь может добавить еще расходы или вернуться к конвертации валют.

Бот должен работать стабильно и отвечать на запросы пользователей в режиме реального времени. Все данные должны храниться в сессии до окончания диалога.

Логика реализованных алгоритмов:

➢ **Импорт необходимых библиотек**

telebot: библиотека для работы с Telegram Bot API.
currency_converter: библиотека для конвертации валют.
types: модуль из telebot для создания типов сообщений и кнопок.

➢ **Инициализация бота и глобальных переменных**

bot = telebot.TeleBot('TOKEN'): создание объекта бота с токеном.
currency = CurrencyConverter(): создание объекта для конвертации валют.
amount = 0, income = 0, expenses = []: глобальные переменные для хранения суммы, дохода и списка расходов.

➢ **Обработчик команды /start**

Функция start(message):
Создает кнопки выбора действия: "Конвертация валют💰" и "Планировщик бюджета📝".
Отправляет сообщение пользователю с предложением выбрать действие.

➢ **Обработчик выбора действия**

Функция handle_choice(call):
Определяет действие на основе нажатой кнопки (конвертация валют или планировщик бюджета).
Вызывает соответствующую функцию для начала выбранного действия.

➢ **Конвертация валют**

Начало конвертации:
Функция start_currency_converter(message):
Отправляет сообщение с просьбой ввести сумму для конвертации.
Регистрирует следующий шаг с функцией summa.
Получение суммы:
Функция summa(message):
Проверяет корректность введенной суммы.
Отправляет сообщение с предложением выбрать пару валют.
Если сумма некорректна, повторяет запрос.
Выбор валюты и конвертация:
Функция callback(call):
Определяет выбранную валютную пару.
Выполняет конвертацию суммы.
Отправляет результат конвертации.
Предлагает действия на выбор: "Продолжить💰" или "Планировщик бюджета📝".
Обработка другой валютной пары:
Функция my_currency(message):
Получает пару валют от пользователя.
Выполняет конвертацию.
Отправляет результат и предлагает действия.

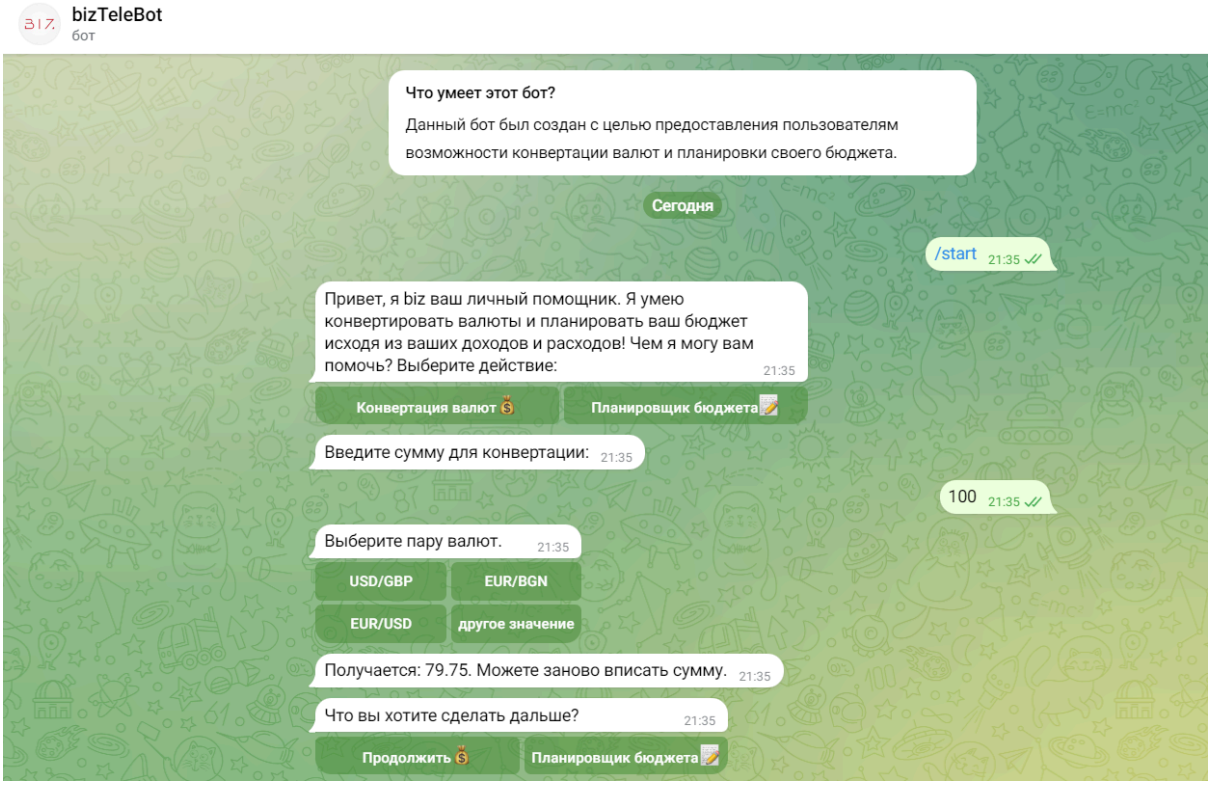
Продолжение конвертации:
Функция continue_currency_converter(message):
Запрашивает новую сумму для конвертации.

➢ **Планировщик бюджета**

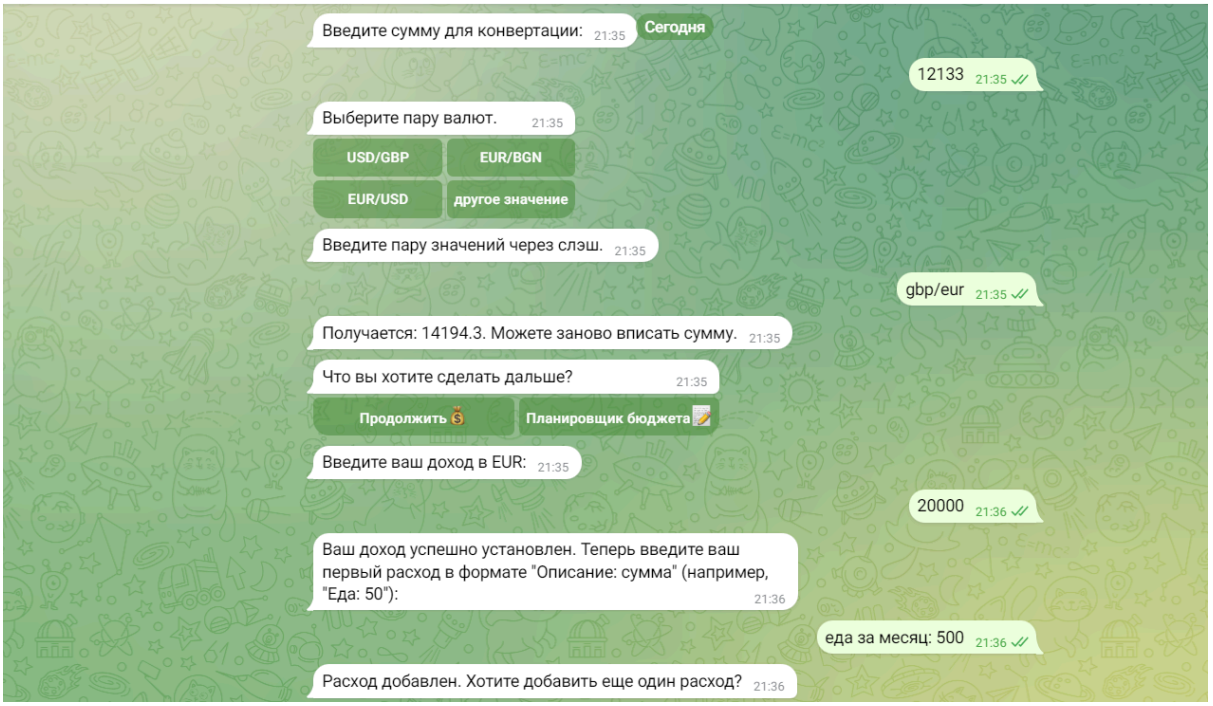
Начало планировщика бюджета:
Функция start_budget_planner(message):
Запрашивает у пользователя доход.
Регистрирует следующий шаг с функцией set_income.
Получение дохода:
Функция set_income(message):
Проверяет корректность введенного дохода.
Запрашивает первый расход.
Если доход некорректен, повторяет запрос.
Добавление расходов:
Функция add_expense(message):
Получает описание и сумму расхода.
Добавляет расход в список.
Предлагает добавить еще один расход или перейти к конвертации валют.
Выбор действия после добавления расхода:
Функция expense_choice(call):
Если выбрано "Да", запрашивает следующий расход.
Если выбрано "Нет", вычисляет баланс.
Вычисление баланса:
Функция calculate_balance(message):
Суммирует расходы.
Вычисляет оставшийся баланс.
Отправляет пользователю отчет о бюджете с советами.
Общие обработчики
Функция callback(call):
Обрабатывает все возможные нажатия кнопок и вызывает соответствующие функции.
Запуск бота
bot.polling(none_stop=True):
Запускает бота в режиме непрерывного опроса, чтобы он всегда был готов обрабатывать сообщения.

Этот бот позволяет пользователю конвертировать валюты и планировать бюджет, предоставляя интуитивно понятный интерфейс с кнопками для выбора действий и пошаговыми инструкциями.

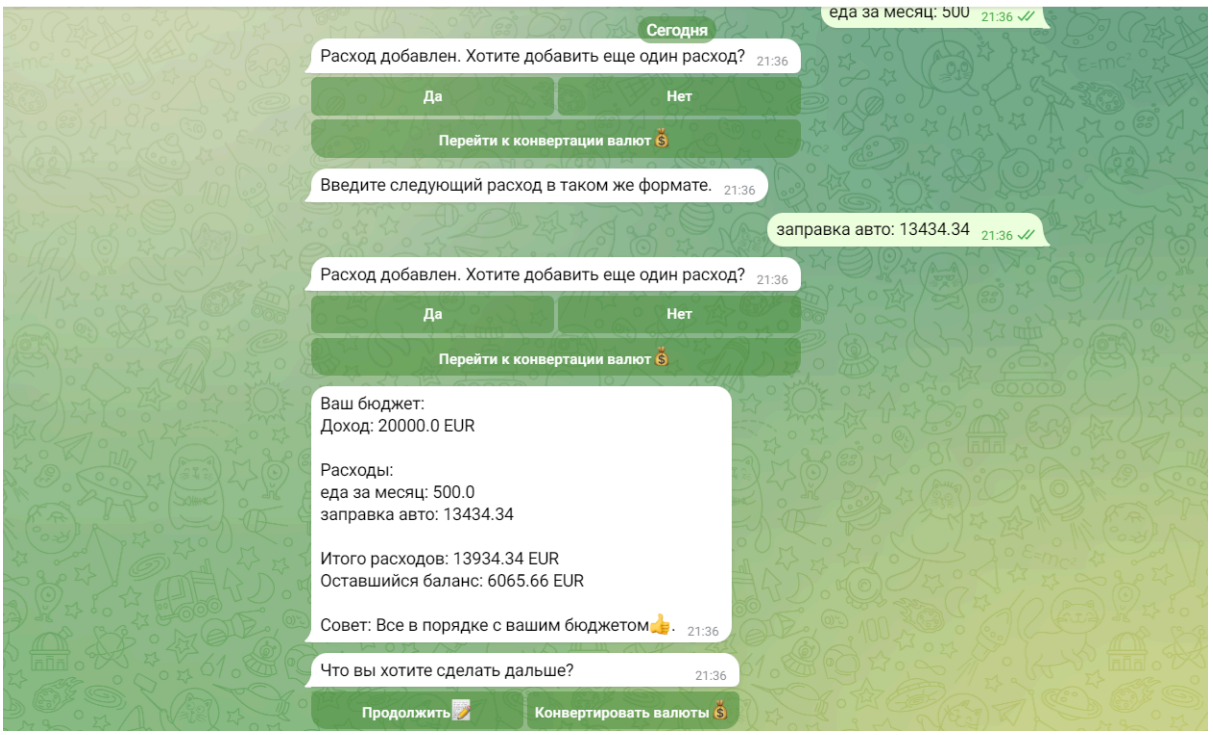
Скрины телеграмм-бота во время взаимодействия с пользователем:



bizTeleBot
бот



bizTeleBot
бот



На данных скриншотах мы видим, как пользователь взаимодействовал с ботом. Для начала после включения бота появилось сообщение “Что умеет этот бот?” и описание что он умеет. После старта пользователь начал с конвертации валют. Он ввел сумму и выбрал пару валют для конвертации.После получения результата он продолжил конвертацию уже другой суммы и других валют. После этого он выбрал “Планировщик бюджета” , ввёл свой бюджет и свои расходы и получил итоговую сумму расходов, оставшийся баланс и совет о его бюджете. После этого он также мог продолжить работу с ботом.

Структуры данных, используемые в мини-приложении:
В данном мини-приложении используются различные структуры данных для хранения и обработки информации.

Простые переменные:

amount и income:

amount - целочисленная переменная для хранения суммы, вводимой пользователем для конвертации валют; income - переменная с плавающей точкой для хранения дохода пользователя в планировщике бюджета.

```
amount = 0
income = 0
```

Списки:

expenses: список для хранения пар "описание: сумма", где каждое описание - строка, а сумма - число с плавающей точкой. Этот список используется для хранения всех расходов пользователя в планировщике бюджета.

```
expenses = []
```

Словари:

currency: объект класса CurrencyConverter, который по сути является словарем, содержащим курсы обмена валют. Используется для конвертации сумм между различными валютами.

```
currency = CurrencyConverter()
```

Описание функционала мини-приложения, строками кода (кусками) и пояснениями:

- ❖ Импорт необходимых библиотек

```
import telebot
from currency_converter import CurrencyConverter
from telebot import types
```

telebot: библиотека для работы с Telegram Bot API.
currency_converter: библиотека для конвертации валют.
types: модуль из telebot для создания типов сообщений и кнопок.

- ❖ Инициализация бота и глобальных переменных

```
bot=telebot.TeleBot('7169851532:AAFJy2K2dWmrXtKOgyF18JtlcFgx8EPHJgc')
currency = CurrencyConverter()
amount = 0
income = 0
expenses = []
```

Создание объекта бота с токеном.
Создание объекта для конвертации валют.
Глобальные переменные для хранения суммы (amount), дохода (income) и списка расходов (expenses).

❖ Обработчик команды /start

```
@bot.message_handler(commands=['start'])
def start(message):
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton('Конвертация валют💰',
callback_data='convert_currency')
    btn2 = types.InlineKeyboardButton('Планировщик бюджета📝',
callback_data='budget_planner')
    markup.add(btn1, btn2)
    bot.send_message(message.chat.id, 'Привет, я biz ваш личный помощник.
Я умею конвертировать валюты и планировать ваш бюджет исходя из
ваших доходов и расходов! Чем я могу вам помочь? Выберите действие:',
reply_markup=markup)
```

Создает кнопки для выбора действия: "Конвертация валют💰" и "Планировщик бюджета📝".
Отправляет сообщение пользователю с предложением выбрать действие.

❖ Обработчик выбора действия

```
@bot.callback_query_handler(func=lambda call: call.data in
['convert_currency', 'budget_planner', 'continue'])
def handle_choice(call):
    if call.data == 'convert_currency':
        start_currency_converter(call.message)
    elif call.data == 'budget_planner':
        start_budget_planner(call.message)
    elif call.data == 'continue':
        continue_currency_converter(call.message)
```

Определяет действие на основе нажатой кнопки (конвертация валют, планировщик бюджета или продолжение конвертации).
Вызывает соответствующую функцию для начала выбранного действия.

❖ Конвертация валют

1. Начало конвертации

```
def start_currency_converter(message):
bot.send_message(message.chat.id, 'Введите сумму для конвертации:')
bot.register_next_step_handler(message, summa)
```

Отправляет сообщение с просьбой ввести сумму для конвертации.
Регистрирует следующий шаг с функцией summa.

2. Получение суммы

```
def summa(message):

    global amount

    try:

        amount = int(message.text.strip())

    except ValueError:

        bot.send_message(message.chat.id, 'Неверный формат, введите сумму.')

        bot.register_next_step_handler(message, summa)

        return



    if amount > 0:

        markup = types.InlineKeyboardMarkup(row_width=2)

        btn1 = types.InlineKeyboardButton('USD/GBP', callback_data='usd/gbp')
```

```
        btn2 = types.InlineKeyboardButton('EUR/BGN', callback_data='eur/bgn')

        btn3 = types.InlineKeyboardButton('EUR/USD', callback_data='eur/usd')

        btn4 = types.InlineKeyboardButton('другое значение',
callback_data='else')

        markup.add(btn1, btn2, btn3, btn4)

        bot.send_message(message.chat.id, 'Выберите пару валют.',
reply_markup=markup)

    else:

        bot.send_message(message.chat.id, 'Число должно быть больше 0.')

        bot.register_next_step_handler(message, summa)
```

Проверяет корректность введенной суммы.
Отправляет сообщение с предложением выбрать пару валют.
Если сумма некорректна, повторяет запрос.

3. Выбор валюты и конвертация

```
@bot.callback_query_handler(func=lambda call: True)

def callback(call):

    if call.data in ['usd/gbp', 'eur/bgn', 'eur/usd']:

        values = call.data.upper().split('/')

        res = currency.convert(amount, values[0], values[1])
```

```
        bot.send_message(call.message.chat.id, f'Получается: {round(res, 2)}. Можете заново вписать сумму.')



        markup = types.InlineKeyboardMarkup(row_width=2)

        btn1 = types.InlineKeyboardButton('Продолжить💰', callback_data='continue')

        btn2 = types.InlineKeyboardButton('Планировщик бюджета📝', callback_data='budget_planner')

        markup.add(btn1, btn2)



        bot.send_message(call.message.chat.id, 'Что вы хотите сделать дальше?', reply_markup=markup)

    elif call.data == 'else':

        bot.send_message(call.message.chat.id, 'Введите пару значений через слэш.')

        bot.register_next_step_handler(call.message, my_currency)

    elif call.data == 'budget_planner':

        start_budget_planner(call.message)

    elif call.data == 'convert_currency':

        start_currency_converter(call.message)
```

```
    elif call.data in ['yes', 'no']:

        expense_choice(call)
```

Определяет выбранную валютную пару.
Выполняет конвертацию суммы.
Отправляет результат конвертации.
Предлагает действия на выбор: "Продолжить💰" или "Планировщик бюджета📝".

4. Обработка другой валютной пары

```
def my_currency(message):

    try:

        values = message.text.upper().split('/')

        res = currency.convert(amount, values[0], values[1])

        bot.send_message(message.chat.id, f'Получается: {round(res, 2)}. Можете заново вписать сумму.')



        markup = types.InlineKeyboardMarkup(row_width=2)

        btn1 = types.InlineKeyboardButton('Продолжить💰', callback_data='continue')

        btn2 = types.InlineKeyboardButton('Планировщик бюджета📝', callback_data='budget_planner')

        markup.add(btn1, btn2)
```

bot.send_message(message.chat.id, 'Что вы хотите сделать дальше?', reply_markup=markup)

except Exception:

bot.send_message(message.chat.id, 'Возникла ошибка. Возможно, введенная валюта не поддерживается. Пожалуйста, введите пару заново.')

bot.register_next_step_handler(message, my_currency)

Получает пару валют от пользователя.
Выполняет конвертацию.
Отправляет результат и предлагает действия.

5. Продолжение конвертации

def continue_currency_converter(message):

bot.send_message(message.chat.id, 'Введите сумму для конвертации:')

bot.register_next_step_handler(message, summa)

Запрашивает новую сумму для конвертации.

Планировщик бюджета

1. Начало планировщика бюджета

```
def start_budget_planner(message):

    bot.send_message(message.chat.id, 'Введите ваш доход в EUR:')

    bot.register_next_step_handler(message, set_income)
```

Запрашивает у пользователя доход.
Регистрирует следующий шаг с функцией set_income.

2. Получение дохода

```
def set_income(message):

    global income

    try:

        income = float(message.text.strip())

        bot.send_message(message.chat.id, 'Ваш доход успешно установлен. Теперь введите ваш первый расход в формате "Описание: сумма" (например, "Еда: 50"):')

        bot.register_next_step_handler(message, add_expense)

    except ValueError:

        bot.send_message(message.chat.id, 'Неверный формат, введите доход в числовом формате.')

        bot.register_next_step_handler(message, set_income)
```

Проверяет корректность введенного дохода.
Запрашивает первый расход.
Если доход некорректен, повторяет запрос.

3. Добавление расходов

```
def add_expense(message):
    global expenses
    try:
        description, amount = message.text.split(':')
        amount = float(amount.strip())
        expenses.append((description.strip(), amount))
        markup = types.InlineKeyboardMarkup(row_width=2)
        btn1 = types.InlineKeyboardButton('Да', callback_data='yes')
        btn2 = types.InlineKeyboardButton('Нет', callback_data='no')
        btn3 = types.InlineKeyboardButton('Перейти к конвертации валют💰', callback_data='convert_currency')
        markup.add(btn1, btn2, btn3)
        bot.send_message(message.chat.id, 'Расход добавлен. Хотите добавить еще один расход?', reply_markup=markup)
    except ValueError:
```

bot.send_message(message.chat.id, 'Неверный формат, введите расход в формате "Описание: сумма" (например, "Еда: 50").')

bot.register_next_step_handler(message, add_expense)

Получает описание и сумму расхода.
Добавляет расход в список.
Предлагает добавить еще один расход или перейти к конвертации валют.

4. Выбор действия после добавления расхода

```
def expense_choice(call):

    if call.data == 'yes':

        bot.send_message(call.message.chat.id, 'Введите следующий расход в таком же формате.')

        bot.register_next_step_handler(call.message, add_expense)

    elif call.data == 'no':

        calculate_balance(call.message)
```

Если выбрано "Да", запрашивает следующий расход.
Если выбрано "Нет", вычисляет баланс.

5. Вычисление баланса

```
def calculate_balance(message):

    total_expenses = sum(amount for description, amount in expenses)

    balance = income - total_expenses
```

```
    advice = 'Все в порядке с вашим бюджетом👍.' if balance >= 0 else 'Ваши расходы превышают доходы, попробуйте сократить расходы📉.'



    expense_details = '\n'.join([f'{description}: {amount}' for description, amount in expenses])

    response = (f'Ваш бюджет:\nДоход: {income} EUR\n\nРасходы:\n{expense_details}\n\nИтого расходов: {total_expenses} EUR\n'

                f'Оставшийся баланс: {balance} EUR\n\nСовет: {advice}')



    bot.send_message(message.chat.id, response)



    markup = types.InlineKeyboardMarkup(row_width=2)

    btn1 = types.InlineKeyboardButton('Продолжить📝', callback_data='budget_planner')

    btn2 = types.InlineKeyboardButton('Конвертировать валюты💰', callback_data='convert_currency')

    markup.add(btn1, btn2)

    bot.send_message(message.chat.id, 'Что вы хотите сделать дальше?', reply_markup=markup)
```

Суммирует расходы.
Вычисляет оставшийся баланс.
Отправляет пользователю отчет о бюджете с советами.

Запуск бота
bot.polling(none_stop=True)
Запускает бота в режиме непрерывного опроса, чтобы он всегда был готов обрабатывать сообщения.

Выводы:
Python известен своей простотой и читаемостью, что делает его идеальным для создания Telegram-ботов. Синтаксис Python интуитивно понятен, что облегчает разработку и поддержку кода.Обширная стандартная библиотека и поддержка сторонних модулей, таких как telebot и currency_converter, позволяют быстро добавлять функциональность. Динамическая типизация ускоряет процесс разработки, хотя требует внимательного отношения к обработке данных.Python позволяет быстро разрабатывать и тестировать приложения благодаря своей простоте и богатому набору инструментов. Активное сообщество разработчиков и хорошо написанная документация делают его отличным выбором для создания ботов и других веб-приложений.