

Geekbrains

**Разработка чат-бота на Python с использованием библиотек для  
обработки естественного языка для решения математических  
задач**

IT-специалист:  
Искусственный интеллект  
Горькова М.В.

Москва  
2024

# СОДЕРЖАНИЕ

<b>СОДЕРЖАНИЕ.....</b>	<b>1</b>
<b>ВВЕДЕНИЕ.....</b>	<b>2</b>
Актуальность темы.....	2
Цель проекта.....	3
Задачи проекта.....	4
Обоснование темы проекта.....	5
<b>ОСНОВНАЯ ЧАСТЬ.....</b>	<b>7</b>
Методология разработки.....	7
Описание инструментов и технологий.....	8
Библиотеки для обработки естественного языка.....	8
Библиотеки для математических вычислений.....	14
Библиотека для разработки бота.....	22
Библиотека для ввода/вывода.....	25
Архитектура чат-бота для решения математических задач.....	26
Разработка чат-бота.....	30
Процесс проектирования.....	30
Процесс разработки чат-бота.....	32
Пример работы чат-бота.....	39
ЗАКЛЮЧЕНИЕ.....	45
Возможности для улучшения и расширения функционала чат-бота.....	46
<b>СПИСОК ЛИТЕРАТУРЫ.....</b>	<b>49</b>

# **ВВЕДЕНИЕ**

С развитием технологий и увеличением интереса к искусственному интеллекту, чат-боты становятся важным инструментом в различных областях, включая образование и помощь в решении математических задач.

Многие студенты и учащиеся сталкиваются с трудностями в понимании и решении математических задач. Традиционные методы обучения не всегда обеспечивают необходимую поддержку. Чат-бот, который может быстро и точно решать математические задачи, способен помочь пользователям получить ответы на свои вопросы.

## **Актуальность темы**

Актуальность темы "Разработка чат-бота на Python с использованием библиотек для обработки естественного языка для решения математических задач" можно рассмотреть с нескольких аспектов:

1) Рост интереса к искусственному интеллекту: В последние годы наблюдается значительный рост интереса к разработке и применению чат-ботов и систем искусственного интеллекта. Чат-боты становятся все более популярными в различных сферах, включая образование, поддержку клиентов и развлечения.

2) Образование и доступность знаний: Чат-боты могут служить полезным инструментом в образовательных целях, помогая учащимся решать математические задачи и объясняя концепции. Это делает обучение более доступным и интерактивным.

3) Обработка естественного языка (NLP): Использование библиотек для обработки естественного языка, таких как NLTK, SpaCy или Transformers, позволяет чат-ботам понимать и интерпретировать запросы пользователей на естественном языке, что значительно улучшает взаимодействие с пользователем.

4) Автоматизация рутинных задач: Чат-боты могут автоматизировать решение рутинных математических задач, освобождая время для более сложных задач и позволяя пользователям сосредоточиться на критическом мышлении и анализе.

5) Интеграция с другими технологиями: Чат-боты могут быть интегрированы с другими технологиями, такими как системы управления обучением (LMS) или платформы для онлайн-обучения, что расширяет их функциональность и возможности.

6) Актуальность в контексте современных вызовов: В условиях быстро меняющегося мира и необходимости дистанционного обучения чат-боты могут стать важным инструментом для поддержки студентов и преподавателей.

Таким образом, разработка чат-бота на Python с использованием библиотек для обработки естественного языка для решения математических задач является актуальной темой, которая сочетает в себе современные технологии, образовательные потребности и возможности автоматизации.

## **Цель проекта**

Целью данной работы является разработка чат-бота, который сможет эффективно обрабатывать запросы пользователей и предоставлять решения математических задач.

1. Создание интерактивного чат-бота: Разработать функционального чат-бота, который сможет взаимодействовать с пользователями и помогать им решать математические задачи на естественном языке.

2. Обеспечение доступности образования: Содействовать обучению и повышению математической грамотности, предоставляя пользователям возможность получать помощь в режиме реального времени.

3. Использование технологий обработки естественного языка (NLP): Применить современные библиотеки и алгоритмы NLP для улучшения понимания запросов пользователей и генерации ответов.

4. Автоматизация решения математических задач: Обеспечить автоматизированное решение различных типов математических задач, включая арифметику, алгебру и геометрию.

### **Задачи проекта**

1. Анализ требований:

- Определить целевую аудиторию и их потребности.
- Составить список типов математических задач, которые бот должен уметь решать.

2. Выбор технологий:

- Определить библиотеки и инструменты для разработки (например, Python, NLTK, SpaCy, SymPy для решения математических задач).
- Выбрать платформу для развертывания чат-бота (например, консоль, Telegram, Discord или веб-платформы).

3. Разработка архитектуры чат-бота:

- Спроектировать архитектуру системы, включая компоненты для обработки входящих сообщений, анализа запросов и генерации ответов.
- Реализовать структуру базы данных (если необходимо) для хранения информации о пользователях и их запросах.

4. Реализация функциональности:

- Разработать модули для обработки естественного языка и распознавания математических задач.
- Реализовать алгоритмы для решения различных типов математических задач.
- Создать интерфейс взаимодействия с пользователем.

5. Тестирование:

- Провести тестирование на разных этапах разработки для выявления и устранения ошибок.

- Оценить точность распознавания запросов и качество решений задач.

#### 6. Сбор обратной связи:

- Запустить бота в тестовом режиме и собрать отзывы пользователей.
- Внедрить улучшения на основе полученной обратной связи.

#### 7. Документация:

- Подготовить документацию по установке, настройке и использованию чат-бота.
- Создать руководство пользователя для конечных пользователей.

#### 8. Поддержка и обновление:

- Обеспечить поддержку пользователей после запуска чат-бота.
- Регулярно обновлять функциональность и улучшать алгоритмы обработки запросов на основе новых данных и технологий.

Эти задачи помогут структурировать процесс разработки чат-бота и обеспечить его успешное внедрение в практику.

## **Обоснование темы проекта**

Я, являясь репетитором по математике, сталкиваюсь постоянно с однотипными задачами, которые отличаются только числами. Мне необходимо сделать бота, который выводит только ответы для разных типов задач. Изначально я сделаю помощника для учеников 9 класса для решения первой части ОГЭ.

Темы первой части ОГЭ (не считая первых пяти задач по картинке):

1. Числа и вычисления
2. Числовые неравенства, координатная прямая
3. Числа, вычисления и алгебраические выражения
4. Уравнения, системы уравнений
5. Статистика, вероятности
6. Графики функций
7. Расчеты по формулам

8. Неравенства, системы неравенств
9. Задачи на прогрессии
10. Треугольники, четырёхугольники, многоугольники и их элементы
11. Окружность, круг и их элементы
12. Площади фигур
13. Фигуры на квадратной решётке
14. Анализ геометрических высказываний

На начальном этапе телеграм-бот будет решать задачи:

1. Решение арифметических выражений
2. Решение уравнений
3. Решение неравенств
4. Решение задач классической вероятности
5. Решение задач арифметической прогрессии
6. Построение графиков

# ОСНОВНАЯ ЧАСТЬ

## Методология разработки

Обоснование выбора инструментов и технологий:

### 1. Язык программирования: Python

- Простота и читаемость: Python известен своей лаконичностью и простотой, что делает его идеальным для быстрого прототипирования и разработки.
- Широкая экосистема библиотек: Python предлагает множество библиотек для обработки естественного языка (NLP), математических вычислений и разработки чат-ботов.
- Поддержка сообщества: Большое сообщество разработчиков обеспечивает доступ к обширной документации и ресурсам, что облегчает решение возникающих проблем.

### 2. Библиотеки для обработки естественного языка (NLP)

- *NLTK (Natural Language Toolkit)*: NLTK — одна из самых популярных библиотек для NLP, предлагающая широкий набор инструментов для обработки текста, включая токенизацию, стемминг, лемматизацию и анализ синтаксиса. Преимущества: Простота использования и наличие обучающих материалов.
- *spaCy*: spaCy ориентирована на производительность и подходит для создания промышленных приложений. Она предлагает эффективные методы для анализа текста и извлечения информации. Преимущества: Быстрота работы и возможность работы с большими объемами данных.

### 3. Библиотеки для математических вычислений

- *SymPy*: SymPy — библиотека для символьной математики, позволяющая выполнять алгебраические операции, решать уравнения и производить упрощение выражений. Преимущества: Возможность получения



аналитических решений и простота интеграции с другими компонентами проекта.

- *NumPy*: NumPy — библиотека для работы с многомерными массивами и матрицами, а также для выполнения математических операций над ними. Преимущества: Высокая производительность при работе с числовыми данными.

- *Matplotlib*: Matplotlib — библиотека на языке программирования Python для визуализации данных двумерной и трёхмерной графикой.

#### 4. Библиотека для разработки бота

- *Python-telegram-bot*: Python-telegram-bot — это популярная библиотека для разработки ботов в мессенджере Telegram на языке Python. Она позволяет легко взаимодействовать с Telegram Bot API и создавать различные типы ботов, от простых до сложных.

## Описание инструментов и технологий

### Библиотеки для обработки естественного языка

Обработка естественного языка (Natural Language Processing, NLP) — это область искусственного интеллекта, которая занимается взаимодействием между компьютерами и человеческим языком. В Python существует множество библиотек, которые упрощают задачи NLP. Вот несколько наиболее популярных и мощных из них:

#### 1. NLTK (Natural Language Toolkit)

- Описание: Одна из самых известных библиотек для обработки естественного языка в Python. Она предоставляет инструменты для работы с текстами, включая токенизацию, стемминг, лемматизацию, POS-теггинг и синтаксический анализ.

- Преимущества:

- Большое количество встроенных корпусов и лексиконов.

- Простота использования и хорошая документация.
- Недостатки:
  - Может быть медленной для больших объемов данных.
  - Некоторые функции требуют значительных ресурсов.

Установка:

```
pip install nltk
```

Пример использования:

```
import nltk

from nltk.tokenize import word_tokenize

# Загрузка необходимых ресурсов

nltk.download('punkt')

text = "Hello, world! This is an example sentence."

tokens = word_tokenize(text)

print(tokens)
```

Вывод:

```
['Hello', ',', 'world', '!', 'This', 'is', 'an', 'example', 'sentence', '.']
```

NLTK (Natural Language Toolkit) предоставляет множество ресурсов для обработки естественного языка. Вот основные категории ресурсов, доступных в NLTK:

- Корпусы текстов: NLTK включает в себя различные корпуса текстов, которые можно использовать для обучения и тестирования моделей:
  - Gutenberg: классические литературные произведения.
  - Brown: многообразие текстов с различными жанрами.
  - Reuters: новости и статьи.
  - Inaugural: тексты инаугурационных речей президентов США.
- Словари и лексические ресурсы
  - WordNet: большой лексический ресурс, который группирует слова по смыслу и показывает их взаимосвязи (синонимы, антонимы и т.д.).

- Word lists: списки слов для различных целей (например, стоп-слова).
- Модели и инструменты
  - Токенизация: функции для разделения текста на слова, предложения и символы.
  - Частеречная разметка (POS tagging): определение частей речи для каждого слова в тексте.
  - Синтаксический анализ: инструменты для построения синтаксических деревьев.
- Стоп-слова: NLTK предоставляет списки стоп-слов для различных языков, которые можно использовать для удаления часто встречающихся, но неинформативных слов из текста.
- Стемминг и лемматизация
  - Стеммеры: алгоритмы для сокращения слов до их корней (например, Porter Stemmer).
  - Лемматизаторы: инструменты для приведения слов к их базовой форме (например, WordNetLemmatizer).
- Модели машинного обучения: NLTK предлагает инструменты для создания и обучения моделей машинного обучения для задач NLP, таких как классификация текста.
- Инструменты визуализации: NLTK включает в себя средства для визуализации данных и результатов анализа, например, построение графиков частоты слов.
- Установка ресурсов: Для работы с этими ресурсами вам может понадобиться загрузить их с помощью команды *nltk.download()*. Например:

```
import nltk
nltk.download('punkt') # Токенизация
nltk.download('averaged_perceptron_tagger') # Частеречная
разметка
nltk.download('wordnet') # WordNet
```

*nlTK.download('stopwords') # Смон-слова*

Эти ресурсы делают NLTK мощным инструментом для обработки и анализа текстовых данных.

## 2. spaCy

- Описание: Современная библиотека для NLP, ориентированная на производительность и использование в реальных приложениях. spaCy поддерживает множество языков и включает в себя модели для различных задач.

- Преимущества:

- Высокая скорость обработки текста.
- Поддержка глубокого обучения и интеграция с другими библиотеками (например, TensorFlow, PyTorch).
- Удобный API и возможность работы с большими объемами данных.

- Недостатки:

- Меньше функций по сравнению с NLTK для некоторых задач.

spaCy — это мощная библиотека для обработки естественного языка, которая предоставляет множество ресурсов и инструментов. Вот основные категории ресурсов, доступных в spaCy:

- Модели языка: spaCy предлагает предобученные модели для различных языков, которые включают:

- Токенизацию: разделение текста на слова и символы.
- Частеречную разметку (POS tagging): определение частей речи для каждого токена.
- Синтаксический анализ: построение зависимостей между словами.
- Лемматизацию: приведение слов к их базовой форме.

- Корпусы и лексические ресурсы

- Word Vectors: предобученные векторы слов, которые помогают в задачах семантического анализа.
- Словари: встроенные лексические ресурсы, такие как списки стоп-слов.

- Инструменты для обработки текста
  - Токенизация: инструменты для разделения текста на токены.
  - Синтаксический анализ: функции для извлечения зависимостей и построения деревьев.
- Named Entity Recognition (NER): spaCy имеет встроенные модели для распознавания именованных сущностей, таких как имена людей, организации, даты и т.д.
- Визуализация
  - Displacy: инструмент для визуализации синтаксических деревьев и именованных сущностей.
- Поддержка пользовательских компонентов: spaCy позволяет легко добавлять пользовательские компоненты в пайплайн обработки текста, что позволяет адаптировать библиотеку под специфические нужды.

Для работы с моделями в spaCy их необходимо загрузить с помощью команды:

```
python -m spacy download <model_name>
```

Например:

```
python -m spacy download en_core_web_sm # Для английского языка
```

Эти ресурсы делают spaCy мощным инструментом для выполнения различных задач обработки естественного языка, таких как анализ текста, извлечение информации и создание приложений на основе NLP.

Установка:

```
pip install spacy
```

```
python -m spacy download en_core_web_sm
```

Пример использования:

```
import spacy
```

```
# Загрузка модели
```

```
nlp = spacy.load("en_core_web_sm")
```

```
text = "Apple is looking at buying U.K. startup for $1 billion"
```

```
doc = nlp(text)
```

```
# Токенизация и извлечение именованных сущностей
for token in doc:
    print(token.text, token.pos_, token.dep_)
print("\nNamed Entities:")
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Вывод:

```
Apple PROPJ nsubj
is AUX aux
looking VERB ROOT
at ADP prep
buying VERB pcomp
U.K. PROPJ pobj
startup NOUN nsubj
for ADP prep
$ SYM quantmod
1 NUM nummod
billion NUM pobj
Named Entities:
Apple ORG
U.K. GPE
$1 billion MONEY
```

Так же есть библиотеки:

- Gensim: Библиотека для тематического моделирования и работы с векторными представлениями слов (word embeddings). Gensim позволяет эффективно обрабатывать большие текстовые корпуса.
- TextBlob: Простая в использовании библиотека для обработки текста. Она предоставляет удобный интерфейс для выполнения различных задач NLP, таких как анализ тональности, перевод и коррекция грамматики.

- Hugging Face Transformers: Библиотека, предоставляющая доступ к предобученным моделям трансформеров для различных задач NLP, таких как генерация текста, перевод, классификация и многие другие.

- Flair: Библиотека для NLP, разработанная на основе PyTorch, предлагает простоту использования и мощные функции для работы с текстами.

Каждая из этих библиотек имеет свои сильные стороны и предназначена для различных задач в области обработки естественного языка. Выбор зависит от ваших требований и предпочтений в работе с текстами. Выбор библиотеки для обработки естественного языка зависит от конкретных задач, объема данных и требований к производительности. NLTK подходит для учебных целей и базового анализа текста, spaCy — для промышленных приложений, а Gensim — для тематического моделирования. Hugging Face Transformers стоит рассмотреть при работе с современными моделями глубокого обучения.

Для нашего проекта будем использовать NLTK и spaCy.

### **Библиотеки для математических вычислений**

#### **1. SymPy**

SymPy — это библиотека для символьных вычислений на языке Python. Она предназначена для выполнения математических операций с символами, что позволяет работать с математическими выражениями в аналитической форме, а не только в числовой.

Основные возможности SymPy:

- Символьные вычисления:
  - Позволяет выполнять операции с символами, такие как сложение, вычитание, умножение и деление.
  - Упрощение выражений и преобразование их в более удобные формы.
- Дифференцирование и интегрирование:
  - Возможность вычисления производных и интегралов различных функций.

- Поддержка как определенных, так и неопределенных интегралов.
- Решение уравнений:
  - Решение алгебраических уравнений и систем уравнений.
  - Решение дифференциальных уравнений (обычных и частных).
- Линейная алгебра:
  - Работа с матрицами и векторами.
  - Операции, такие как определители, собственные значения и собственные векторы.
- Тригонометрические функции и упрощения:
  - Работа с тригонометрическими функциями, их преобразованиями и упрощениями.
- Графическое представление:
  - Возможность визуализации математических функций и выражений с помощью встроенных графических инструментов.
- Поддержка различных форматов вывода:
  - Возможность вывода результатов в LaTeX, MathML и других форматах, что удобно для публикаций и отчетов.
- Интеграция с другими библиотеками:
  - SymPy может работать в связке с другими библиотеками Python, такими как NumPy и Matplotlib, для выполнения численных расчетов и визуализации.

Применение в различных областях:

- Наука и техника: Используется в физике для моделирования физических процессов, в инженерии для анализа систем.
- Образование: Широко применяется в учебных заведениях для обучения математике, физике и смежным дисциплинам.
- Исследования: Используется в научных исследованиях для анализа данных и разработки новых алгоритмов.

Установка:

SymPy можно установить через pip:



*pip install sympy*

Примеры использования:

```
import sympy as sp  
# Определяем символы  
x = sp.symbols('x')  
# Создаем математическое выражение  
expr = sp.sin(x)**2 + sp.cos(x)**2  
# Упрощаем выражение  
simplified_expr = sp.simplify(expr)  
# Вычисляем производную  
derivative = sp.diff(sp.sin(x), x)  
# Вычисляем интеграл  
integral = sp.integrate(sp.sin(x), x)  
# Решаем уравнение  
solution = sp.solve(sp.Eq(sp.sin(x), 0), x)  
print(f"Упрощенное выражение: {simplified_expr}")  
print(f"Производная: {derivative}")  
print(f"Интеграл: {integral}")  
print(f"Решение уравнения: {solution}")
```

SymPy — мощный инструмент для тех, кто работает с математикой на высоком уровне. Его возможности делают его незаменимым как для студентов, так и для профессионалов в различных областях науки и техники.

## 2. NumPy

NumPy (Numerical Python) — это фундаментальная библиотека для численных вычислений на языке Python. Она предоставляет поддержку многомерных массивов и матриц, а также множество функций для выполнения математических операций над этими массивами. NumPy является основой для

многих других библиотек в экосистеме Python, таких как SciPy, Pandas и Matplotlib.

Основные возможности NumPy:

- Массивы (ndarray):
  - NumPy предоставляет объект *ndarray* (n-dimensional array), который позволяет хранить данные в многомерных массивах. Это более эффективно по памяти и быстрее по сравнению с обычными списками Python.
- Векторизация:
  - NumPy позволяет выполнять операции над массивами без использования циклов, что значительно ускоряет вычисления. Это достигается за счет применения векторизированных функций.
- Универсальные функции (ufuncs):
  - NumPy включает в себя множество математических функций, которые могут быть применены к массивам. Эти функции работают элемент-wise, что позволяет выполнять операции над всеми элементами массива одновременно.
- Индексация и срезы:
  - NumPy поддерживает мощную индексацию и срезы, что позволяет легко извлекать и изменять данные в массивах.
- Линейная алгебра:
  - Библиотека предоставляет функции для выполнения операций линейной алгебры, таких как умножение матриц, вычисление определителей и собственных значений.
- Статистические функции:
  - NumPy включает в себя множество функций для выполнения статистических расчетов, таких как среднее, медиана, стандартное отклонение и т.д.
- Работа с файлами:

- NumPy позволяет загружать и сохранять массивы в различных форматах, включая текстовые файлы и бинарные форматы.
- Интеграция с другими библиотеками:
  - NumPy является основой для многих других библиотек, таких как SciPy (научные вычисления), Pandas (анализ данных) и Matplotlib (визуализация данных).

Установка: NumPy можно установить через pip:

```
pip install numpy
```

Примеры использования:

```
import numpy as np
# Создание одномерного массива
a = np.array([1, 2, 3, 4])
print("Одномерный массив:", a)
# Создание двумерного массива
b = np.array([[1, 2], [3, 4]])
print("Двумерный массив:\n", b)
# Векторизация: сложение двух массивов
c = a + 10
print("Сложение с 10:", c)
# Индексация
print("Первый элемент массива a:", a[0])
# Срезы
print("Элементы с 1 по 3:", a[1:3])
# Операции линейной алгебры
d = np.dot(b, b) # Умножение матриц
print("Умножение матриц:\n", d)
# Статистические функции
mean_value = np.mean(a)
print("Среднее значение массива a:", mean_value)
# Создание массива из равномерно распределенных чисел
```

```
e = np.linspace(0, 1, 5) # 5 чисел от 0 до 1  
print("Равномерно распределенные числа:", e)  
# Сохранение и загрузка массива  
np.save('my_array.npy', a) # Сохранение  
loaded_array = np.load('my_array.npy') # Загрузка  
print("Загруженный массив:", loaded_array)
```

NumPy — это мощный инструмент для научных и инженерных вычислений на Python. Его возможности позволяют эффективно работать с большими объемами данных и выполнять сложные математические операции. Библиотека широко используется в научных исследованиях, анализе данных и машинном обучении, что делает ее незаменимой для многих специалистов.

### 3. Matplotlib

*Matplotlib* — это популярная библиотека для визуализации данных в Python. Она предоставляет мощные инструменты для создания статических, анимационных и интерактивных графиков. Matplotlib особенно полезна в научных исследованиях, анализе данных и машинном обучении.

Основные возможности Matplotlib:

- Создание графиков:
  - Matplotlib позволяет создавать различные типы графиков, включая линейные графики, столбчатые диаграммы, круговые диаграммы, гистограммы, плотности, 3D-графики и многие другие.
- Настройка графиков:
  - Вы можете настраивать практически все аспекты графиков: цвета, размеры, шрифты, метки осей, заголовки, легенды и т.д.
- Поддержка нескольких графиков:
  - Matplotlib позволяет размещать несколько графиков на одной фигуре с помощью подграфиков (subplots).
- Интерактивные графики:

- С помощью Matplotlib можно создавать интерактивные графики, которые позволяют пользователям взаимодействовать с данными (например, масштабирование и панорамирование).
- Экспорт графиков:
  - Графики можно сохранять в различных форматах, таких как PNG, PDF, SVG и т.д., что удобно для публикации и отчетов.
- Интеграция с другими библиотеками:
  - Matplotlib хорошо интегрируется с NumPy и Pandas, что делает его удобным для визуализации данных из этих библиотек.

Основные компоненты Matplotlib:

- Figure: Основной контейнер для всех элементов графика.
- Axes: Область, где располагается график (ось X и Y).
- Artist: Все элементы, которые отображаются на графике (линии, текст, метки и т.д.).

Установка: Matplotlib можно установить через pip:

```
pip install matplotlib
```

Примеры использования:

```
import matplotlib.pyplot as plt
import numpy as np
# Пример 1: Линейный график
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.figure(figsize=(10, 5)) # Установка размера фигуры
plt.plot(x, y, label='sin(x)', color='blue') # Построение линии
plt.title('Линейный график')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show() # Отображение графика
```

```

# Пример 2: Столбчатая диаграмма
categories = ['A', 'B', 'C', 'D']
values = [4, 7, 1, 8]
plt.bar(categories, values, color='orange')
plt.title('Столбчатая диаграмма')
plt.xlabel('Категории')
plt.ylabel('Значения')
plt.show()

# Пример 3: Гистограмма
data = np.random.randn(1000) # Генерация случайных данных
plt.hist(data, bins=30, color='green', alpha=0.7)
plt.title('Гистограмма')
plt.xlabel('Значение')
plt.ylabel('Частота')
plt.show()

# Пример 4: Круговая диаграмма
sizes = [15, 30, 45, 10]
labels = ['A', 'B', 'C', 'D']
colors = ['gold', 'lightcoral', 'lightskyblue', 'yellowgreen']
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=140)
plt.title('Круговая диаграмма')
plt.axis('equal') # Чтобы круг был кругом
plt.show()

```

Matplotlib — это мощный инструмент для визуализации данных в Python. Его гибкость и широкие возможности настройки позволяют создавать высококачественные графики для научных публикаций и отчетов. С помощью Matplotlib можно легко исследовать данные и представлять результаты анализа в наглядной форме.

## Библиотека для разработки бота

*Python-telegram-bot* — это популярная библиотека для разработки ботов в мессенджере Telegram на языке Python. Она позволяет легко взаимодействовать с Telegram Bot API и создавать различные типы ботов, от простых до сложных.

Основные возможности:

- Удобный интерфейс: Библиотека предоставляет высокоуровневый интерфейс для работы с API, что упрощает разработку.
- Обработка сообщений: Легко обрабатывать текстовые сообщения, команды, кнопки и другие элементы интерфейса.
- Поддержка Webhook и Polling: Можно использовать как Webhook для получения обновлений, так и Polling для опроса сервера.
- Инлайн-режим: Поддержка инлайн-команд и кнопок, позволяющих пользователям взаимодействовать с ботом без необходимости вводить команду.
- Файлы и медиа: Возможность отправки и получения изображений, видео, документов и других медиафайлов.
- Расширяемость: Легко добавлять новые функции и команды, а также интегрировать с другими библиотеками и API.

Установка: Чтобы установить библиотеку, используется `pip`:

```
pip install python-telegram-bot
```

Пример использования:

```
from telegram import Update  
from telegram.ext import Updater, CommandHandler, CallbackContext  
def start(update: Update, context: CallbackContext) -> None:  
    update.message.reply_text('Привет! Я ваш бот.')  
def main():  
    updater = Updater("YOUR_TOKEN", use_context=True)  
    dispatcher = updater.dispatcher  
    dispatcher.add_handler(CommandHandler("start", start))
```

```
updater.start_polling()
updater.idle()
if __name__ == '__main__':
    main()
```

Библиотека `python-telegram-bot` — это мощный инструмент для создания ботов в Telegram, который позволяет разработчикам легко реализовывать различные функции и взаимодействовать с пользователями.

### **Библиотека для работы с регулярными выражениями**

Библиотека `re` в Python — это стандартная библиотека для работы с регулярными выражениями. Регулярные выражения (regex) позволяют выполнять сложные поиски и манипуляции с текстом, такие как поиск, замена и валидация строк.

Основные функции библиотеки `re`

- `re.match(pattern, string):`
  - Проверяет, соответствует ли начало строки заданному шаблону.
  - Возвращает объект совпадения, если найдено совпадение, иначе — `None`.
- `re.search(pattern, string):`
  - Ищет по всей строке первое совпадение с заданным шаблоном.
  - Возвращает объект совпадения или `None`.
- `re.findall(pattern, string):`
  - Находит все неперекрывающиеся совпадения с шаблоном в строке.
  - Возвращает список всех найденных совпадений.
- `re.finditer(pattern, string):`
  - То же, что и `findall`, но возвращает итератор объектов совпадения вместо списка.
- `re.sub(pattern, repl, string):`
  - Заменяет все вхождения шаблона на заданную строку.
  - Возвращает новую строку с заменами.



- *re.split(pattern, string):*
  - Разделяет строку по шаблону.
  - Возвращает список подстрок.

Специальные символы: В регулярных выражениях есть множество специальных символов, которые помогают задавать шаблоны:

- . — любой символ, кроме новой строки.
- ^ — начало строки.
- \$ — конец строки.
- \* — 0 или более вхождений предыдущего символа.
- + — 1 или более вхождений предыдущего символа.
- ? — 0 или 1 вхождение предыдущего символа.
- {n} — ровно n вхождений.
- {n,} — n или более вхождений.
- {n,m} — от n до m вхождений.
- [] — набор символов (например, [a-z]).
- | — логическое "ИЛИ".

Пример использования:

```
import re

# Пример строки
text = "Hello, my email is example@example.com and my phone number
is 123-456-7890."

# Поиск email
email_pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
email_matches = re.findall(email_pattern, text)
print("Emails found:", email_matches)

# Поиск номера телефона
phone_pattern = r'\d{3}-\d{3}-\d{4}'
phone_matches = re.findall(phone_pattern, text)
print("Phone numbers found:", phone_matches)

# Замена email на [EMAIL]
```

```
masked_text = re.sub(email_pattern, '[EMAIL]', text)
print("Masked text:", masked_text)
```

Библиотека *re* предоставляет мощные инструменты для работы с текстом и регулярными выражениями. Она полезна для задач, связанных с парсингом строк, валидацией данных и текстовой обработкой.

### Библиотека для ввода/вывода

Библиотека *io* в Python предоставляет интерфейсы для работы с потоками ввода и вывода. Она включает в себя как текстовые, так и бинарные потоки, а также классы для работы с файловыми объектами. Основная цель *io* — обеспечить более гибкий и эффективный способ работы с данными.

Основные компоненты библиотеки *io*:

- Потоки ввода-вывода\*\*:
  - *io.TextIOWrapper*: Обертка для текстовых потоков, которая позволяет работать с текстом (строками) в разных кодировках.
  - *io.BytesIO*: Виртуальный бинарный поток, который работает с байтовыми данными в памяти.
  - *io.StringIO*: Виртуальный текстовый поток, который работает со строками в памяти.
- Работа с файлами\*\*:
  - Открытие файлов через встроенную функцию *open()*, которая возвращает объект *TextIOWrapper* или *BufferedReader/BufferedWriter*, в зависимости от режима открытия.

Библиотека *io* предоставляет мощные средства для работы с потоками ввода и вывода, что делает её полезной для обработки данных как в текстовом, так и в бинарном формате. Она позволяет эффективно управлять памятью и работать с данными в реальном времени, что особенно важно в контексте веб-разработки, обработки данных и других приложений.

# Архитектура чат-бота для решения математических задач

## 1. Общая структура чат-ботов.

Чат-бот должен состоять из нескольких ключевых компонентов, взаимодействующих друг с другом:

- Клиентская часть: Интерфейс для пользователя (может быть веб-приложением, мессенджером и т.д.).
- Серверная часть: Обработка запросов и логика работы бота.
- База данных: Хранение данных пользователей и истории взаимодействий.
- Модули обработки: Библиотеки для NLP и математических вычислений.

В нашем случае будет реализован простейший бот, который не будет содержать БД, а вся серверная часть будет реализована в Visual Studio Code без использования фреймворков Flask или FastAPI. Бот будет работать при запущенной программе.

## 2. Компоненты системы

### 2.1. Клиентская часть

- Интерфейс: Пользователь может взаимодействовать с ботом через текстовые сообщения.
- Протокол общения: Использование API для отправки и получения сообщений (Telegram Bot API).

### 2.2. Серверная часть

- Обработчик сообщений: Принимает текстовые сообщения от пользователя и передает их в модуль NLP для анализа.

### 2.3. Модуль обработки естественного языка (NLP)

- Токенизация: Разделение текста на отдельные слова и символы.
- Анализ намерений: Определение, что именно хочет пользователь (например, решить уравнение или получить помощь с задачей).

- Извлечение сущностей: Определение математических операций и чисел в запросе.

#### 2.4. Модуль математических вычислений

- Символьные вычисления: Использование SymPy для решения уравнений и выполнения алгебраических операций.
- Численные вычисления: Использование NumPy для работы с массивами и числовыми данными.

#### 2.5. Логирование вместо БД

- Хранение данных в файле .csv: История взаимодействий пользователей с ботом.

### 3. Взаимодействие компонентов

- Пользователь отправляет сообщение через клиентский интерфейс (через Telegram).
- Сервер принимает сообщение и передает его в модуль NLP.
- Модуль NLP анализирует сообщение, определяет намерение и извлекает необходимые данные.
- Математический модуль выполняет вычисления и возвращает результат.
- Сервер формирует ответ и отправляет его обратно пользователю через клиентский интерфейс.

### 4. Примерный поток данных

Пользователь -> Клиентская часть -> Сервер -> NLP (NLTK/spaCy) ->  
Математические вычисления (SymPy/NumPy) -> Логирование -> Ответ  
пользователю

### 5. Безопасность и обработка ошибок

- Валидация ввода: Проверка входящих данных на корректность перед обработкой.
- Обработка исключений: Информирование пользователя о возможных проблемах.

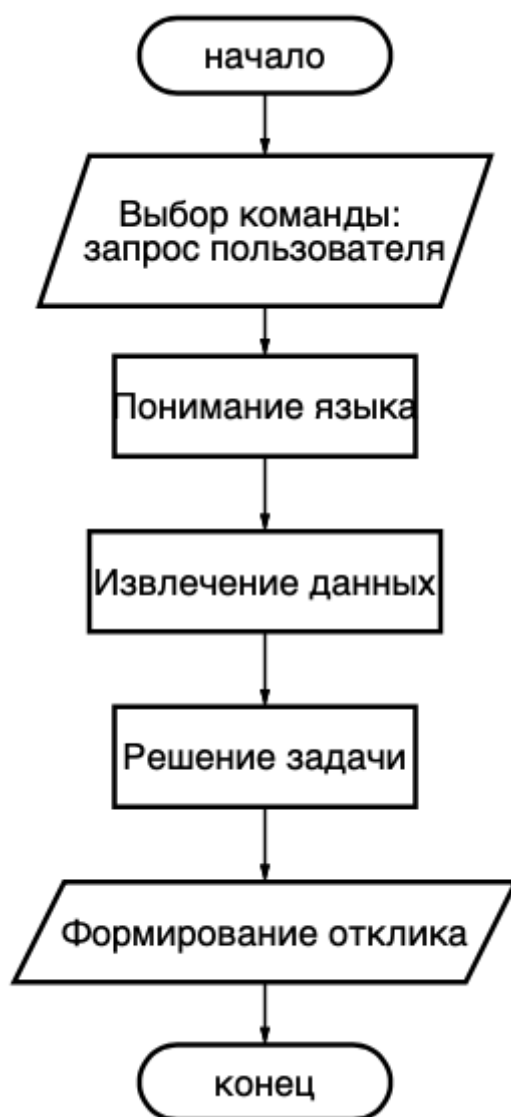


Рис. 1 - Блок-схема структуры чат-бота

Эта архитектура обеспечивает модульность, что позволяет легко обновлять или заменять отдельные компоненты, а также масштабировать систему по мере необходимости. Чат-бот будет удобным инструментом для пользователей, желающих решать математические задачи с помощью простого текстового интерфейса.

# Разработка чат-бота

## Процесс проектирования

Процесс проектирования Telegram чат-бота для решения математических задач включает несколько ключевых этапов. Каждый из этих этапов помогает структурировать работу и обеспечить успешную реализацию проекта.

### 1. Определение требований

#### 1.1. Функциональные требования

- Пользователь должен иметь возможность задавать математические задачи (уравнения, примеры).

- Бот должен уметь распознавать математические операции и числа.
- Бот должен предоставлять решения задач и объяснения.
- Возможность сохранять историю запросов и ответов.

#### 1.2. Нефункциональные требования

- Быстрая обработка запросов (отвечать в течение нескольких секунд).

- Удобный интерфейс общения через Telegram.

### 2. Выбор технологий

#### 2.1. Языки и библиотеки

- Язык программирования: Python.
- Библиотеки для работы с Telegram: `python-telegram-bot`.
- Библиотеки для NLP: NLTK, `spaCy`.
- Библиотеки для математических вычислений: `SymPy`, `NumPy`.

#### 2.2. Система управления базами данных (будет использовано в будущем)

- - SQLite для простоты или PostgreSQL для более сложных решений.

### 3. Архитектура системы

#### 3.1. Общая архитектура

- Клиентская часть: Telegram API.
- Серверная часть: Visual Studio Code.
- Модули обработки: NLP и математические вычисления.
- Логирование: Хранение истории взаимодействий.

### 3.2. Взаимодействие компонентов

Определить, как будут взаимодействовать различные модули.

- Пользователь отправляет сообщение через клиентский интерфейс (через Telegram).
- Сервер принимает сообщение и передает его в модуль NLP.
- Модуль NLP анализирует сообщение, определяет намерение и извлекает необходимые данные.
- Математический модуль выполняет вычисления и возвращает результат.
- Сервер формирует ответ и отправляет его обратно пользователю через клиентский интерфейс.

## 4. Разработка прототипа

4.1. Создание минимально жизнеспособного продукта (MVP):  
Реализовать базовую функциональность: обработка текстовых запросов и возврат простых решений.

4.2. Тестирование MVP: Провести тестирование на предмет корректности работы и выявления ошибок.

## 5. Разработка основной функциональности

5.1. Реализация NLP модуля: Обработка входящих сообщений и извлечение математических выражений.

5.2. Реализация математического модуля: Решение уравнений и выполнение вычислений.

6. Тестирование и отладка (выполнено не будет, так как нет необходимых компетенций, должен делать тестировщик)

6.1. Модульное тестирование: Проверка отдельных компонентов на корректность работы.

- 6.2. Интеграционное тестирование: Проверка взаимодействия между модулями.
- 6.3. Пользовательское тестирование: Тестирование с реальными пользователями для получения обратной связи.
- 7. Поддержка и обновление
  - 7.1. Мониторинг работы бота: Настройка логирования для отслеживания.
  - 7.2. Обновление функциональности: Регулярно добавлять новые функции на основе обратной связи от пользователей.

Проектирование чат-бота — это итеративный процесс, который требует внимания к деталям на каждом этапе. Важно собирать обратную связь от пользователей и адаптироваться к их потребностям, чтобы создать полезный и эффективный инструмент для решения математических задач через Telegram.

### **Процесс разработки чат-бота**

#### **1. Описание функциональности чат-бота**

##### **1.1. Описание математических функций в телеграм-боте**

###### **1.1.1. Решение арифметических выражений:**

Арифметические операции, которые должен выполнять бот:

- Сложение
- Вычитание
- Умножение
- Деление

###### **1.1.2. Решение арифметических уравнений:**

Бот должен решать уравнения общего вида: справа в выражении должен быть ноль.

###### **1.1.3. Решение неравенств:**

Бот должен решать неравенства общего вида: справа в выражении должен быть ноль.

###### **1.1.4. Решение задач классической вероятности:**



Классическое определение вероятности основано на понятии равновозможности исходов.

Вероятностью называется отношение количества исходов, благоприятствующих данному событию, к общему числу равновозможных исходов.

$$P = \frac{m}{n}$$

#### 1.1.5. Решение задач арифметической прогрессии:

Последовательность чисел (членов прогрессии), в которой каждое число, начиная со второго, получается из предыдущего добавлением к нему постоянного числа (шага, или разности прогрессии):

Определение	$a_{n+1} = a_n + d$
Разность	$d = a_{n+1} - a_n$
Формула n-го члена	$a_n = a_1 + (n - 1) \cdot d$
Сумма n первых членов	$S_n = \frac{a_1 + a_n}{2} \cdot n \quad S_n = \frac{2a_1 + (n - 1) \cdot d}{2} \cdot n$
Свойство	$a_n = \frac{a_{n+1} + a_{n-1}}{2}$

Рис. 2 - Формулы арифметической прогрессии

#### 1.1.6. Построение графиков

Геометрический образ функции

#### 1.2. Создание чат-бота

- Создаем чат-бот через *BotFatherBot* через команду: */newbot*.
- Назовем нового бота: *diploma\_gb\_bot*. Токен скрыт для безопасности.

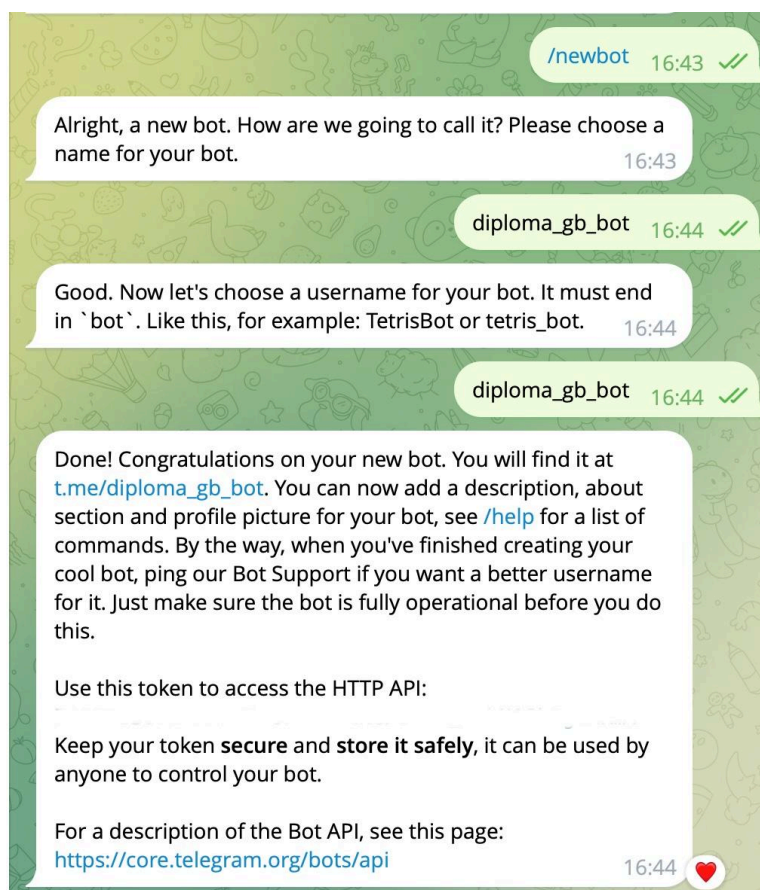


Рис. 3 - Создание телеграм-бота

### 1.3. Основные функции в программе

1.3.1. Приветственное сообщение (/hi): “Hi, “имя””.

1.3.2. Команда “помощь” (/help): Выводит все функции, которые есть на выбор.

1.3.3. Решение арифметических выражений: Бот принимает и обрабатывает арифметические выражения, используя библиотеки для вычислений (SymPy).

Данная функция будет принимать текстовые запросы на решение математических задач и использовать библиотеку sympy для вычислений.

- **Обработка ввода:** Чат-бот принимает ввод пользователя и использует spaCy для анализа текста. Он извлекает числовые выражения и знаки операций (+, -, \*, /).
- **Вычисление результата:** После извлечения математического выражения оно передается в функцию *evaluate\_expression*, где используется библиотека *sympy* для вычисления результата.

- Вывод результата: Бот возвращает результат пользователю.

1.3.4. Решение арифметических уравнений: Бот распознает уравнения и предоставляет пользователю решение. Алгоритм, который используется для решения арифметических уравнений:

Функция для решения арифметических уравнений с использованием библиотеки *sympy* для математических вычислений и *nltk* для обработки естественного языка (NLP).

Эта функция будет принимать текстовый запрос, извлекать уравнение и решать его:

- Функция *solve\_equation*: Эта функция принимает строку уравнения, преобразует её в выражение *sympy* и решает его. Она поддерживает как линейные, так и квадратные уравнения.
- Функция *interpret\_query*: Эта функция обрабатывает текстовый запрос, токенизирует его и использует регулярное выражение для поиска уравнения. Она возвращает первое найденное уравнение.
- Функция *main*: Эта функция объединяет всё вместе. Она принимает текстовый запрос, извлекает уравнение и решает его.

Функция будет извлекать уравнение и предоставлять его решение.

1.3.5. Решение неравенств: Опишите подходы к решению неравенств, включая метод интервалов и графический анализ.

Для решения неравенств в Python можно использовать библиотеку *sympy*, которая предоставляет мощные инструменты для работы с математическими выражениями. Функция принимает строку с неравенством и решает его:

- Импорт библиотеки: Мы импортируем библиотеку *sympy*.
- Определение функции: Функция *solve\_inequality* принимает строку с неравенством.
- Определение переменной: Мы определяем переменную *x* как символ.
- Преобразование строки: С помощью *sympify* мы преобразуем строку в математическое выражение.

- Решение неравенства: Используем *solve\_univariate\_inequality*, чтобы найти решение неравенства.
- Проверка типа: После преобразования строки в неравенство проверяется, является ли оно экземпляром *sp.Rel* (что означает, что это действительно неравенство). Если нет, выбрасывается *ValueError*.
- Обработка исключений:
  - *sp.SympifyError*: обрабатывает ошибки, возникающие при преобразовании строки в математическое выражение.
  - *ValueError*: обрабатывает ошибки, связанные с некорректным типом выражения.
  - Общий *Exception*: отлавливает любые другие неожиданные ошибки.
- Возврат результата: Функция возвращает решение.

1.3.6. Решение задач классической вероятности: Создание функции для решения задач по теории вероятности с использованием обработки естественного языка (NLP) содержит базовую структуру, которая будет извлекать информацию из текстового запроса и решать простые задачи по теории вероятности с использованием библиотеки *nltk*.

- Функция *extract\_probability\_info*: Эта функция ищет числа в запросе, которые могут представлять количество благоприятных исходов и общее количество исходов. Она использует регулярное выражение для извлечения чисел.
- Функция *calculate\_probability*: Эта функция принимает количество благоприятных исходов и общее количество исходов, затем вычисляет вероятность события.
- Функция *main*: Она объединяет всё вместе, извлекает информацию из запроса и вычисляет вероятность.
- Проверка на отрицательные значения: В функции *main* добавлена проверка на отрицательные значения для *events* и *total\_outcomes*. Если они отрицательные, возвращается соответствующее сообщение.

- Обработка исключений: Блоки *try-except* добавлены вокруг основного кода в функции *main*, чтобы обрабатывать возможные ошибки, такие как ошибки преобразования данных (*ValueError*) и любые другие неожиданные исключения.

Функция будет извлекать данные о вероятностях и предоставлять результат.

1.3.7. Решение задач арифметической прогрессии: Бот может находить сумму, *n*-й член прогрессии.

Создание функции для решения задач на арифметическую прогрессию с использованием методов обработки естественного языка (NLP) может включать в себя интерпретацию текстовых запросов пользователей и извлечение необходимых параметров для вычислений. Функция на Python использует библиотеку *spaCy* для обработки текста.

- Функция *extract\_parameters*: Эта функция обрабатывает текстовый ввод пользователя и извлекает значения первого члена (*a*), разности (*d*) и номера члена (*n*). Она использует *spaCy* для анализа текста.
- Функции *nth\_term* и *sum\_of\_n\_terms*: Эти функции аналогичны тем, что были описаны ранее и выполняют вычисления для арифметической прогрессии.

- Функция *nth\_term*: Эта функция принимает первый член прогрессии *a*, разность *d* и номер члена *n*. Она вычисляет *n*-й член по формуле:

$$a_n = a + d * (n - 1)$$

- Функция *sum\_of\_n\_terms*: Эта функция принимает тот же набор параметров и вычисляет сумму первых *n* членов по формуле:

$$a_n = \frac{2 * a + d * (n - 1)}{2} * n$$

- Функция *main*: Здесь происходит взаимодействие с пользователем. Пользователь вводит текстовый запрос, из которого извлекаются

параметры. Если параметры успешно извлечены, программа вычисляет и выводит результаты.

1.3.8. Построение графиков: Бот генерирует графики по запросу пользователя с помощью библиотек, используемых для визуализации функций и уравнений. Для построения графика функции с использованием Python и NLP (Natural Language Processing) можно использовать библиотеки *matplotlib* для построения графиков и *sympy* для работы с математическими выражениями. Функция обрабатывает текстовые сообщения и пытается построить график на основе введенной пользователем функции. Если функция некорректная или возникает ошибка при построении графика, бот отправляет соответствующее сообщение.

- Импорт библиотек: Мы импортируем необходимые библиотеки для построения графиков (*matplotlib*) и для работы с математическими выражениями (*sympy*).
- Определение функции: Функция *plot\_function* принимает строку с описанием функции и строит её график, сохраняя его в байтовом потоке.
- Преобразование строки в выражение: С помощью *sympify* мы преобразуем строку в математическое выражение.
- Создание массива значений: Используем *np.linspace* для создания массива значений  $X$  в заданном диапазоне.
- Вычисление значений функции: Мы вычисляем значения функции для каждого значения  $X$ .
- Построение графика: Используем *matplotlib* для визуализации графика.

1.3.9. Логирование: Функция *log* предназначена для ведения журнала взаимодействий пользователей с ботом. Она позволяет отслеживать, кто и какие сообщения отправляет, что может быть полезно для анализа поведения пользователей или отладки бота.

## Пример работы чат-бота

Текст кода представлен в приложении.

Запускаем программу в *Microsoft Visual Studio*.

Открываем чат-бота *diploma\_gb\_bot*.

Для начала работы необходимо ввести команду */start*.

Рассмотрим работу каждой существующей команды:

1. */hi*

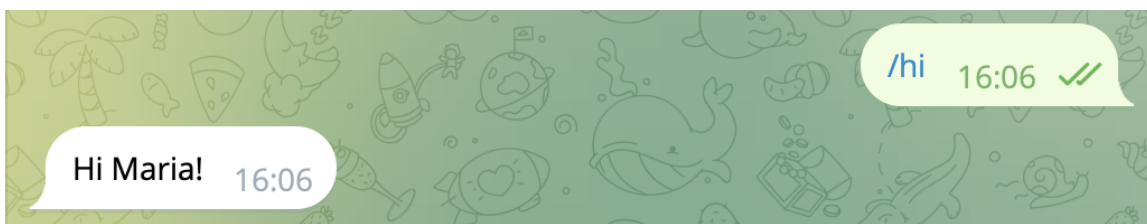


Рис. 4 - Пример работы команды */hi*

2. */help*



Рис. 5 - Пример работы команды */help*

3. */calculations*

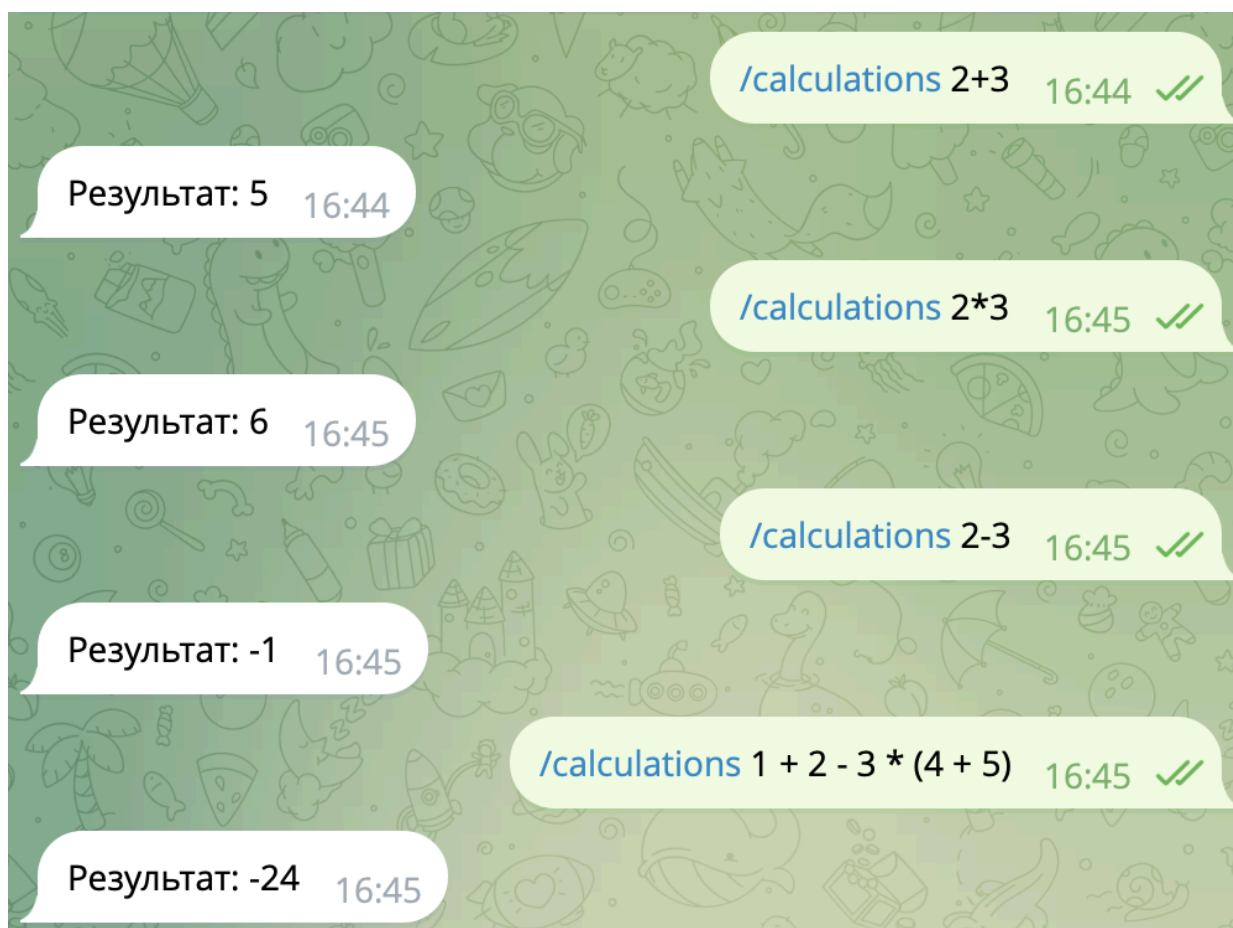


Рис. 5 - Пример работы команды `/calculations` при корректных входных данных (только числа и знаки)

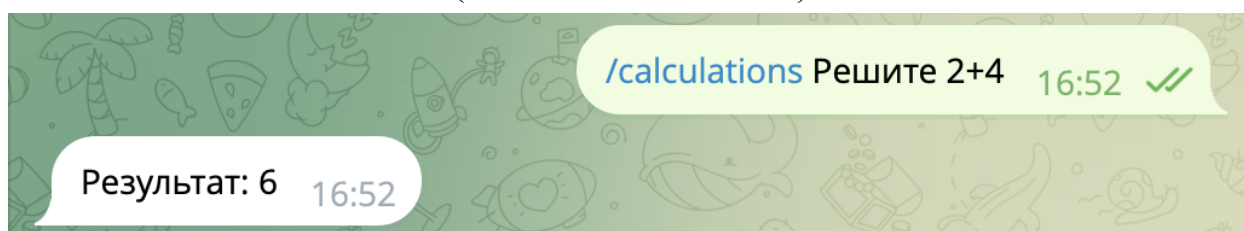


Рис. 6 - Пример работы команды `/calculations` при корректных входных данных (в выражении есть слова)

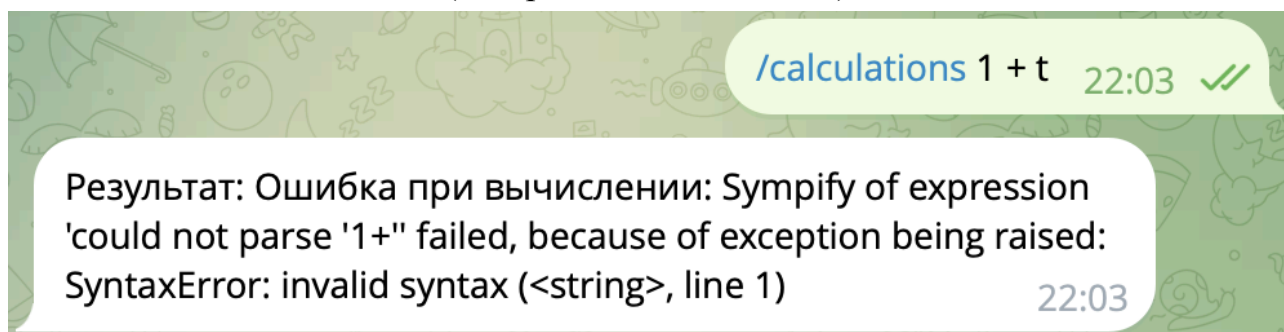


Рис. 7 - Пример работы команды `/calculations` при некорректных входных данных



#### 4. */equations*

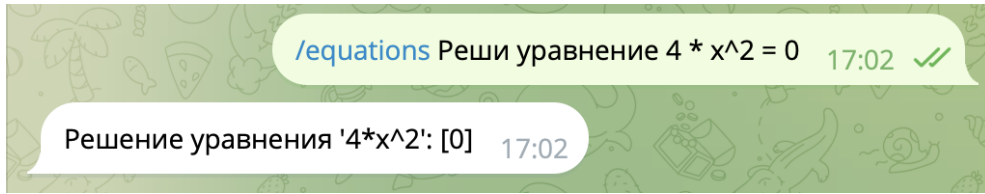


Рис. 8 - Пример работы команды */equations* при корректных входных данных

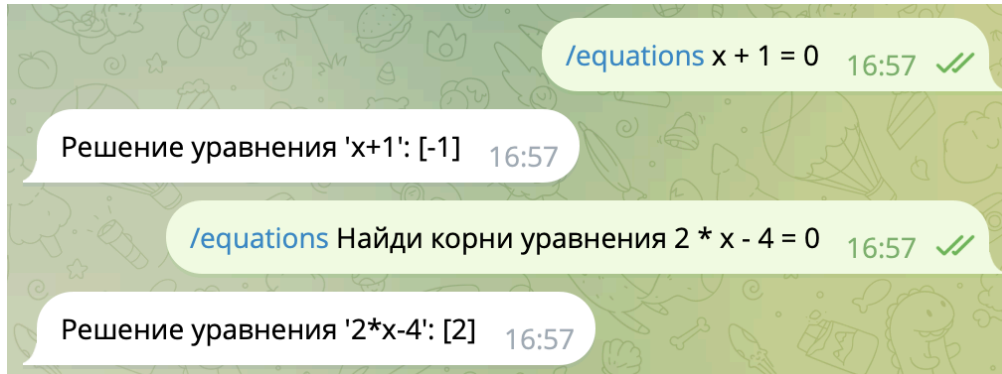


Рис. 9 - Пример работы команды */equations* при корректных входных данных

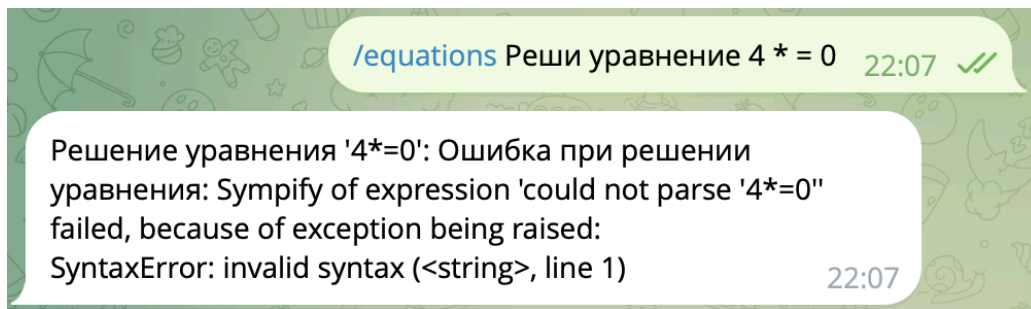


Рис. 10 - Пример работы команды */equations* при некорректных входных данных

#### 5. */inequalities*

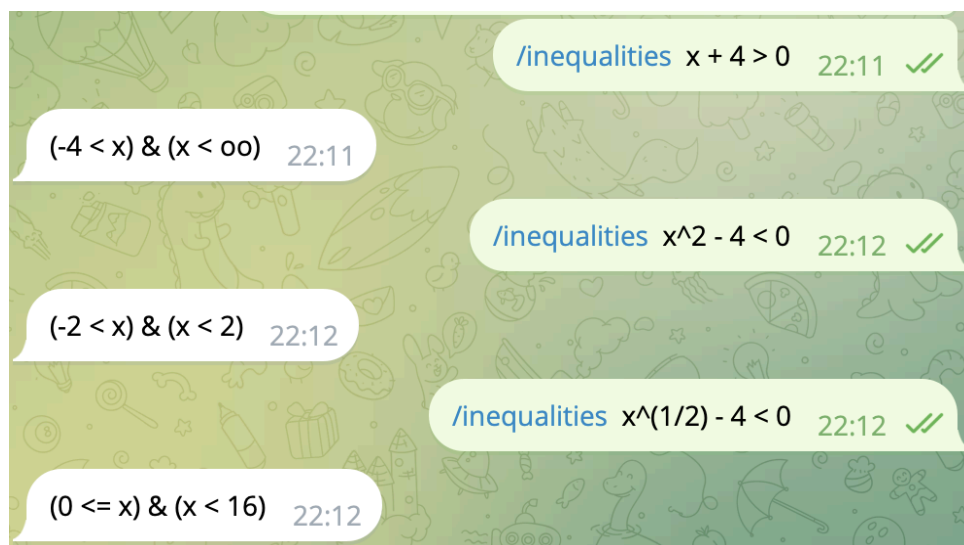


Рис. 11 - Пример работы команды */inequalities* при корректных входных данных

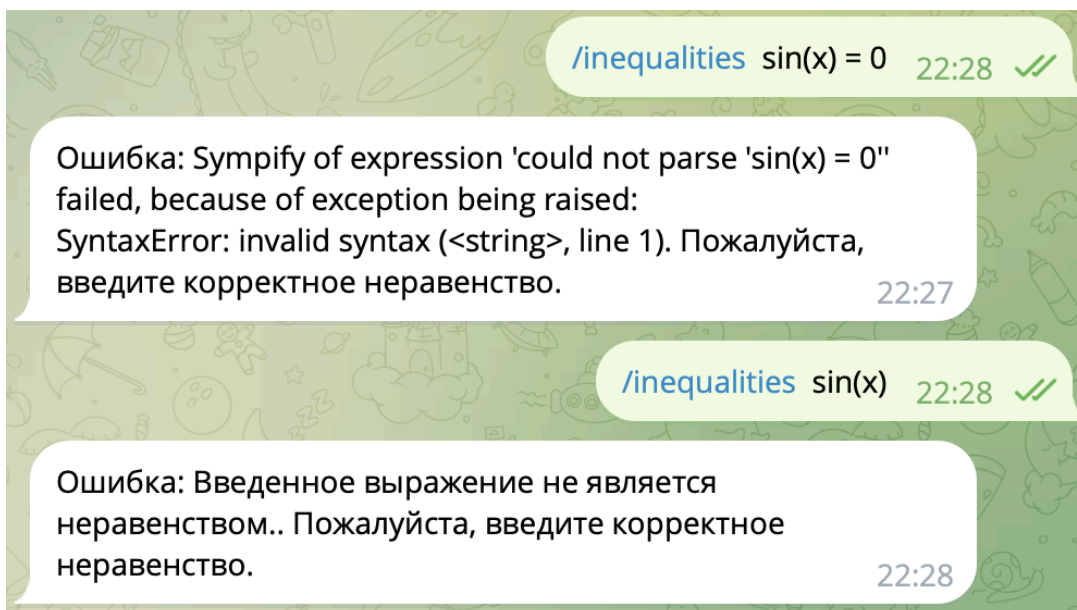


Рис. 11 - Пример работы команды */inequalities* при некорректных входных данных

#### 6. */progressions*

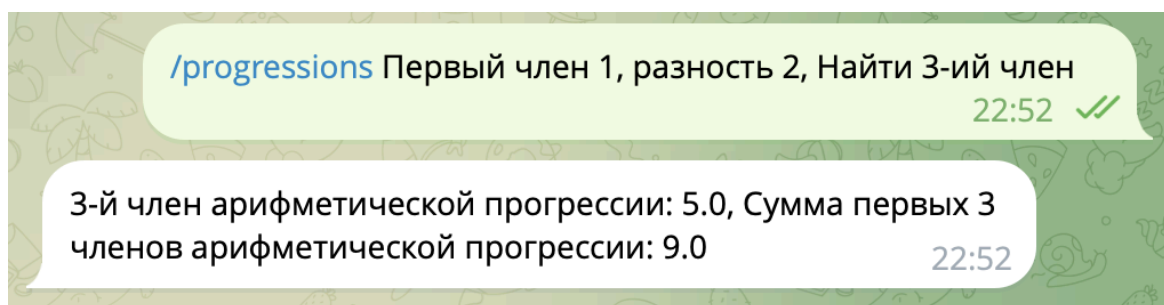


Рис. 12 - Пример работы команды */progressions* при корректных входных данных

#### 7. */probabilities*

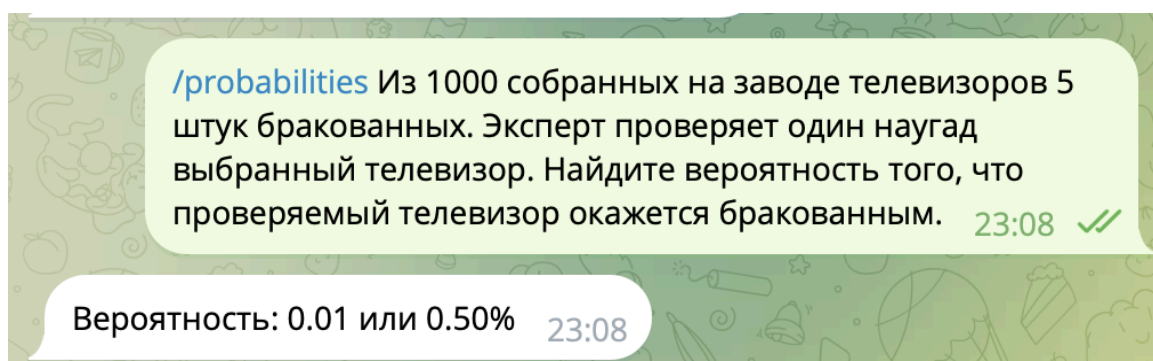


Рис. 13 - Пример работы команды */probabilities* при корректных входных данных

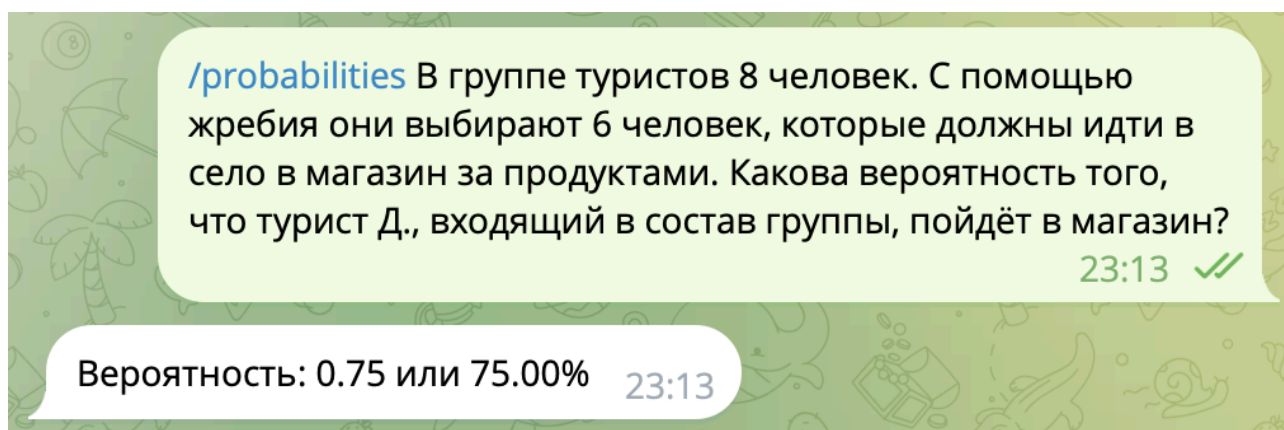


Рис. 14 - Пример работы команды */probabilities* при корректных входных данных

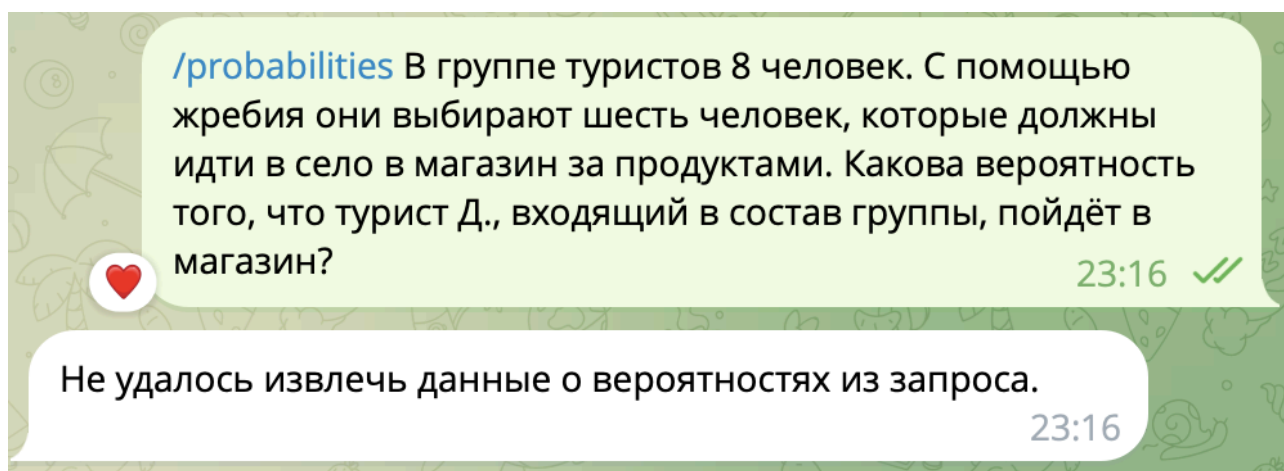


Рис. 15 - Пример работы команды */probabilities* при некорректных входных данных

## 8. */plots*

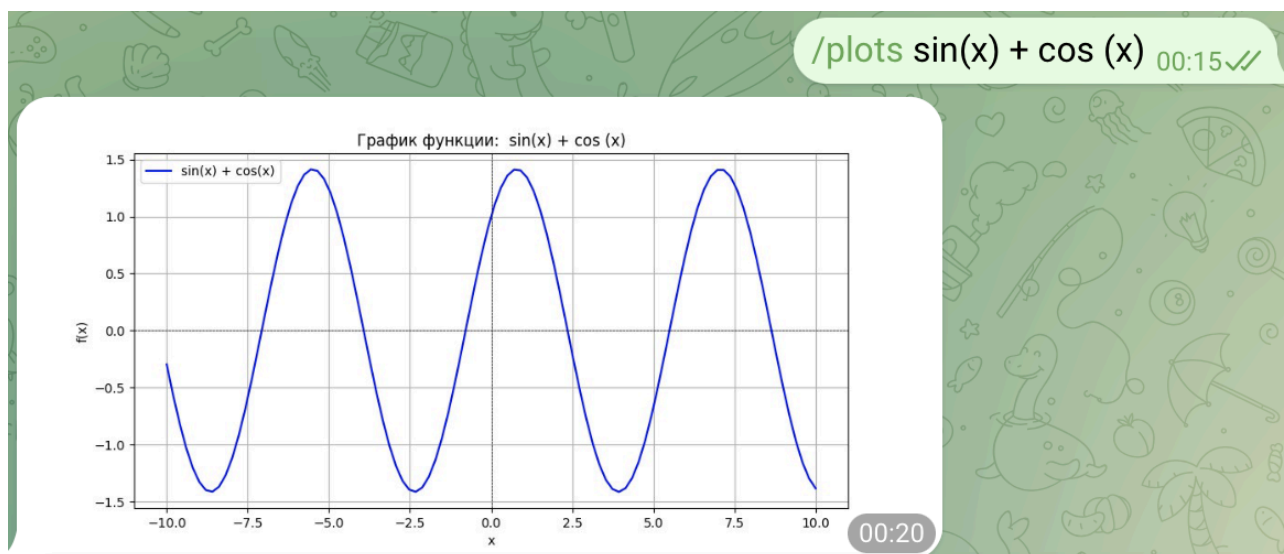


Рис. 16 - Пример работы команды */plots* при корректных входных данных

## 9. *log.csv*

logs.csv

```
1 Maria, 362635593, /hi
2 Maria, 362635593, /help
3 Maria, 362635593, /calculations
4 Maria, 362635593, /calculations 2+3
5 Maria, 362635593, /calculations 2*3
6 Maria, 362635593, /calculations 2/3
7 Maria, 362635593, /calculations 4/2
8 Maria, 362635593, /calculations 2-3
9 Maria, 362635593, /calculations 4//2
10 Maria, 362635593, /calculations 2+3
11 Maria, 362635593, /calculations 2*3
12 Maria, 362635593, /calculations 2-3
13 Maria, 362635593, /calculations 1 + 2 - 3 * (4 + 5)
14 Maria, 362635593, /calculations Решите 2+4
15 Maria, 362635593, /equations  $x + 1 = 0$ 
16 Maria, 362635593, /equations Найди корни уравнения  $2 * x - 4 = 0$ 
17 Maria, 362635593, /equations Реши уравнение  $x**2 - 4 = 0$ 
18 Maria, 362635593, /equations Реши уравнение  $x^2 - 4 = 0$ 
19 Maria, 362635593, /equations Реши уравнение  $x**2 - 4 = 0$ 
20 Maria, 362635593, /equations Реши уравнение  $4 * x^2 = 0$ 
21 Maria, 362635593, /calculations 1 + 2 - 3 *
```

Рис. 16 - log.csv

## ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы на тему "Разработка чат-бота на Python с использованием библиотек для обработки естественного языка для решения математических задач" была успешно реализована система, способная эффективно решать широкий спектр математических задач. Основные функции чат-бота включают:

1. Решение арифметических выражений — бот способен выполнять базовые арифметические операции, что позволяет пользователям быстро получать результаты без необходимости ручных вычислений.

2. Решение уравнений — реализован алгоритм, который позволяет пользователям вводить уравнения и получать точные решения, что значительно упрощает процесс обучения и самопроверки.

3. Решение неравенств — бот может обрабатывать неравенства, предоставляя пользователям возможность изучать их свойства и решения.

4. Решение задач классической вероятности — реализация этой функции открывает новые горизонты для пользователей, интересующихся статистикой и вероятностными расчетами.

5. Решение задач арифметической прогрессии — бот предоставляет возможность анализировать и решать задачи, связанные с последовательностями, что полезно для студентов и школьников.

6. Построение графиков — визуализация данных является важным аспектом в математике, и наш чат-бот предоставляет пользователям возможность видеть графическое представление решений.

Полученные результаты подтверждают, что использование библиотек для обработки естественного языка в сочетании с мощными математическими функциями Python позволяет создать удобный инструмент для обучения и решения математических задач.

Практическое значение работы заключается в том, что разработанный чат-бот может быть использован как в образовательных учреждениях для помощи студентам, так и в качестве самостоятельного инструмента для всех желающих улучшить свои математические навыки.

В дальнейшем, перспективы развития проекта могут включать расширение функционала чат-бота за счет добавления новых типов задач, улучшения обработки естественного языка для повышения точности распознавания запросов пользователей и интеграции с другими образовательными платформами.

Лично для меня данный проект стал ценным опытом в области программирования и разработки интеллектуальных систем. Я получил глубокие знания о работе с библиотеками Python, а также о принципах обработки естественного языка, что, безусловно, будет полезно в моей будущей карьере.

Таким образом, разработка чат-бота не только достигла поставленных целей, но и открыла новые возможности для дальнейших исследований и разработок в области автоматизации учебного процесса.

## **Возможности для улучшения и расширения функционала чат-бота**

1. Расширение математических возможностей
  - 1.1. Поддержка различных типов уравнений: Добавить возможность решать более сложные уравнения.
  - 1.2. Системы уравнений: Реализовать решение систем линейных уравнений.
  - 1.3. Неравенства: Расширить функционал для работы с неравенствами высших степеней.
2. Расширение возможностей по вероятности
  - 2.1. Комбинаторика: Включить задачи на сочетания и перестановки.

- 2.2. Сложные вероятностные задачи: Реализовать возможность решения задач с использованием теорем Байеса, условной вероятности и т.д.
- 3. Работа с прогрессиями
  - 3.1. Геометрическая прогрессия: Добавить возможность решения задач на геометрическую прогрессию.
  - 3.2. Поддержка различных типов сложных задач: Добавить возможность решать более сложные задач.
- 4. Графическое представление
  - 4.1. Интерактивные графики: Использовать библиотеки для создания интерактивных графиков, которые позволят пользователю изменять параметры.
  - 4.2. 3D графики: Добавить возможность строить трехмерные графики для функций нескольких переменных.
- 5. Улучшение взаимодействия с пользователем
  - 5.1. Пошаговое решение: Добавить возможность показывать шаги решения, чтобы пользователь мог видеть процесс.
  - 5.2. Подсказки и объяснения: Реализовать систему подсказок и объяснений по каждому шагу решения.
- 6. Обработка текстовых задач
  - 6.1. Анализ текста: Разработать алгоритмы для извлечения данных из текстовых задач, что позволит решать их автоматически.
  - 6.2. Обработка единиц измерения: Включить поддержку различных единиц измерения (например, метры, километры, секунды и т.д.).
- 7. Интеграция с другими сервисами
  - 7.1. API для внешних расчетов: Интегрироваться с API сторонних сервисов для выполнения сложных математических расчетов.

- 7.2. Поддержка различных языков: Реализовать многоязычность, чтобы бот мог взаимодействовать с пользователями на разных языках.
- 8. Обучение и адаптация
  - 8.1. Машинное обучение: Использовать алгоритмы машинного обучения для адаптации к стилю общения пользователя и улучшения качества ответов.
  - 8.2. История взаимодействий: Сохранять историю запросов пользователя для анализа и улучшения рекомендаций.
- 9. Обратная связь и улучшение
  - 9.1. Система отзывов: Внедрить механизм сбора отзывов от пользователей для постоянного улучшения функционала.
  - 9.2. Обновления и новые функции: Регулярно добавлять новые функции на основе запросов пользователей.

Эти улучшения могут значительно повысить функциональность чат-бота и сделать его более полезным для пользователей.



## СПИСОК ЛИТЕРАТУРЫ

1. Официальная документация Python: Python 3 Documentation (<https://docs.python.org/3/library/io.html>)
2. Официальная документация: [NLTK Documentation](<https://www.nltk.org/>)
3. Официальная документация: [spaCy Documentation](<https://spacy.io/>)
4. Официальная документация: [SymPy Documentation](<https://docs.sympy.org/latest/index.html>)
5. Официальная документация: [NumPy Documentation](<https://numpy.org/doc/stable/>)
6. Официальная документация: [re — Regular Expression Operations](<https://docs.python.org/3/library/re.html>)
7. Официальная документация: [io — Core Tools for Working with Streams](<https://docs.python.org/3/library/io.html>)
8. Официальная документация: [python-telegram-bot] (<https://python-telegram-bot.org/>)

# ПРИЛОЖЕНИЕ

## 1. main.py

```
from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler, ContextTypes
from bot_commands import *

app = ApplicationBuilder().token("7057785091:AAFY746rZSq9lхGP69Bw_6mbU1JbJgCxRm4").build()

app.add_handler(CommandHandler("hi", hi_command))
app.add_handler(CommandHandler("help", help_command))
app.add_handler(CommandHandler("equations", equations_command))
app.add_handler(CommandHandler("calculations", calculations_command))
app.add_handler(CommandHandler("inequalities", inequalities_command))
app.add_handler(CommandHandler("plots", plots_command))
app.add_handler(CommandHandler("probabilities", probabilities_command))
app.add_handler(CommandHandler("progressions", progressions_command))
app.run_polling()
# Работа до прерывания
app.updater.idle()
```

## 2. bot\_commands.py

```
from telegram import Update, InputFile
from telegram.ext import ApplicationBuilder, CommandHandler, ContextTypes
from spy import *
import equations
import calculations
import inequalities
import plots
import progressions
import probabilities

# Приветствие
async def hi_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    log(update, context)
    await update.message.reply_text(f'Hi {update.effective_user.first_name}!')

# Команда помощь, знать какие команды есть
async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    log(update, context)
    await update.message.reply_text(f'/hi\n/help\n/equations\n/calculations\n/inequalities\n/plots\n/progressions\n/probabilities')
```

```

# работа с уравнениями
async def equations_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    log(update, context)
    msg = update.message.text
    await update.message.reply_text(equations.main(msg))

# работа с арифметическими действиями
async def calculations_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    log(update, context)
    msg = update.message.text
    await update.message.reply_text(calculations.process_input(msg))

# работа с неравенствами
async def inequalities_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    # Включаем логирование
    log(update, context)
    msg = update.message.text
    await update.message.reply_text(inequalities.solve_inequality(msg))

# работа с графиками
async def plots_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    log(update, context)
    msg = update.message.text
    #buf = plot_function(user_input)
    await update.send_photo(photo=InputFile(plots.checking_message(msg),
filename='function_plot.png'))
    #await update.message.reply_text(plots.plot_function(msg, x_range=(-10, 10)))

# работа с арифметическими прогрессиями
async def progressions_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    log(update, context)
    msg = update.message.text
    await update.message.reply_text(progressions.main(msg))

# работа с классической вероятностью
async def probabilities_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    log(update, context)
    msg = update.message.text
    await update.message.reply_text(probabilities.main(msg))

```

### 3. calculations.py

```

import spacy
import sympy as sp

```

```

# Загрузка модели spaCy
nlp = spacy.load("en_core_web_sm")

# Функция для обработки математических выражений
def evaluate_expression(expression):
    try:
        # Используем sympy для решения выражений
        result = sp.sympify(expression)
        return result
    except Exception as e:
        return f"Ошибка при вычислении: {str(e)}"

# Функция для обработки ввода пользователя
def process_input(user_input):
    # Применяем NLP для извлечения математического выражения
    doc = nlp(user_input)
    expression = ""

    for token in doc:
        # Ищем числовые выражения и знаки операций
        if token.is_digit or token.text in "+-*/()":
            expression += token.text

    if expression:
        result = evaluate_expression(expression)
        return f"Результат: {result}"
    else:
        return "Не удалось распознать математическое выражение."

```

#### 4. equations.py

```

import re
import sympy as sp

def solve_equation(equation_str):
    """Решает линейные и квадратные уравнения."""
    try:
        x = sp.symbols('x')
        # Преобразуем строку в выражение sympy
        equation = sp.sympify(equation_str)

        # Решаем уравнение
        solutions = sp.solve(equation, x)
        return solutions
    except Exception as e:
        return f"Ошибка при решении уравнения: {e}"

def extract_equation(query):

```

```

"""Извлекает уравнение из текстового запроса."""

# Регулярное выражение для поиска уравнений
equation_pattern = f'([-+]?\d*\.*?x?\s*[-+*/=^]\s*\d+|\d+\s*[-+*/=^]\s*\d*\.*?x?) '
# Ищем уравнение в запросе
matches = re.findall(equation_pattern, query.replace(" ", ""))
print(matches)

if matches:
    #if matches[1] == '=0':
        return matches[0] # Берем первое найденное уравнение
return None

def main(query):
    """Основная функция для обработки запроса и решения уравнения."""
    equation_str = extract_equation(query)
    print(equation_str)

    if equation_str:
        solution = solve_equation(equation_str)
        return f"Решение уравнения '{equation_str}': {solution}"
    else:
        return "Уравнение не найдено в запросе."

```

## 5. inequalities.py

```

import sympy as sp

def solve_inequality(query):
    # удаляем команду
    inequality_str = query.replace('/inequalities', '').strip()

    # Определяем переменную
    x = sp.symbols('x')

    try:
        # Преобразуем строку в математическое неравенство
        inequality = sp.sympify(inequality_str)

        # Проверяем, является ли выражение неравенством
        if not isinstance(inequality, sp.Rel):
            raise ValueError("Введенное выражение не является неравенством.")

        # Решаем неравенство
        solution = sp.solve_univariate_inequality(inequality, x)

        return f'{solution}'

    except (sp.SympifyError, ValueError) as e:

```

```

        return f'Ошибка: {str(e)}. Пожалуйста, введите корректное неравенство.'
    except Exception as e:
        return f'Неизвестная ошибка: {str(e)}. Пожалуйста, проверьте введенные данные.'

```

## 6. plots.py

```

import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
from io import BytesIO

# Обработка текстовых сообщений
def checking_message(user_input):
    # Проверяем корректность введенной функции
    try:
        buf = plot_function(user_input)
        return buf
    except Exception as e:
        return (f'Ошибка при построении графика: {str(e)}')

def plot_function(query, x_range=(-10, 10), num_points=100):
    # удаляем саму команду
    function_str = query.replace('/plots', '')

    # Определяем переменную
    x = sp.symbols('x')

    # Преобразуем строку в математическое выражение
    function = sp.sympify(function_str)

    # Создаем массив значений x
    x_vals = np.linspace(x_range[0], x_range[1], num_points)

    # Вычисляем значения функции
    y_vals = [function.evalf(subs={x: val}) for val in x_vals]

    # Строим график
    plt.figure(figsize=(10, 5))
    plt.plot(x_vals, y_vals, label=str(function), color='blue')
    plt.title(f'График функции: {function_str}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.axhline(0, color='black', lw=0.5, ls='--')
    plt.axvline(0, color='black', lw=0.5, ls='--')
    plt.grid()
    plt.legend()
    plt.show()

    # Сохраняем график в байтовый поток

```

```

buf = BytesIO()
plt.savefig(buf, format='png')
buf.seek(0)
plt.close()
return buf

```

## 7. probabilities.py

```

import nltk
import ssl
import re
from nltk.tokenize import word_tokenize

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

nltk.download('punkt')

def extract_probability_info(query):
    """Извлекает информацию о вероятностях из текстового запроса."""
    # Пример паттерна для поиска вероятностей
    pattern = r'(\d+)%?'
    matches = re.findall(pattern, query)

    if matches:
        return sorted([int(match) for match in matches])

    return None

def calculate_probability(events, total_outcomes):
    """Вычисляет вероятность события."""
    if total_outcomes == 0:
        return "Общее количество исходов не может быть нулевым."

    probability = events / total_outcomes
    return probability

def main(query):
    """Основная функция для обработки запроса и вычисления вероятности."""
    try:
        probabilities = extract_probability_info(query)

        if probabilities and len(probabilities) == 2:
            events, total_outcomes = probabilities
            if events < 0 or total_outcomes < 0:

```

```

        return "Количество событий и общее количество исходов должны быть неотрицательными."

    probability = calculate_probability(events, total_outcomes)
    return f"Вероятность: {probability:.2f} или {probability * 100:.2f}%"
else:
    return "Не удалось извлечь данные о вероятностях из запроса."

except ValueError as ve:
    return f"Ошибка преобразования данных: {ve}"
except Exception as e:
    return f"Произошла ошибка: {e}"

```

## 8. progressions.py

```

import spacy

# Загрузка модели NLP
nlp = spacy.load("en_core_web_sm")

def extract_parameters(text):
    """Извлекает параметры a, d и n из текстового запроса."""
    doc = nlp(text)
    a, d, n = None, None, None

    for token in doc:
        if token.text.lower() == 'первый' and token.nbor(2).text.lower() == 'член':
            try:
                a = float(token.nbor(2).text)
            except ValueError:
                print(f"Ошибка: '{token.nbor(2).text}' не является числом.")
        elif token.text.lower() == 'разность':
            try:
                d = float(token.nbor(1).text)
            except ValueError:
                print(f"Ошибка: '{token.nbor(1).text}' не является числом.")
        elif token.text.lower() == 'найти':
            try:
                n = int(token.nbor(1).text)
            except ValueError:
                print(f"Ошибка: '{token.nbor(1).text}' не является целым числом.")

    return a, d, n

def nth_term(a, d, n):
    """Вычисляет n-й член арифметической прогрессии."""
    try:
        return a + (n - 1) * d
    except TypeError:

```



```

        print("Ошибка: Параметры a, d и n должны быть числами.")

def sum_of_n_terms(a, d, n):
    """Вычисляет сумму первых n членов арифметической прогрессии."""
    try:
        return (n / 2) * (2 * a + (n - 1) * d)
    except TypeError:
        print("Ошибка: Параметры a, d и n должны быть числами.")

def main(user_input):
    a, d, n = extract_parameters(user_input)

    if a is not None and d is not None and n is not None:
        nth = nth_term(a, d, n)
        sum_n = sum_of_n_terms(a, d, n)
        return f"{n}-й член арифметической прогрессии: {nth}, Сумма первых {n} членов арифметической прогрессии: {sum_n}"
    else:
        print("Не удалось извлечь параметры. Пожалуйста, убедитесь, что ваш запрос корректен.")

```

## 9. spy.py

```

from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler, ContextTypes
from bot_commands import *

def log(update: Update, context: ContextTypes.DEFAULT_TYPE):
    file = open('logs.csv', 'a')
    file.write(f'{update.effective_user.first_name}, {update.effective_user.id}, {update.message.text}\n')
    file.close()

```