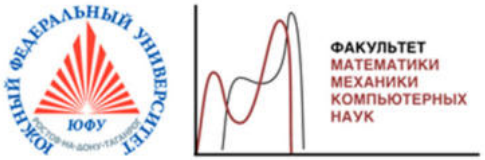
Многоуровневая архитектура управления LLM-ассистентом в предметно-ограниченной образовательной среде

В.М. Абрамян

Южный федеральный университет,
Институт математики, механики
и компьютерных наук им. И. И. Воровича

vikabramyan@sfedu.ru



Идея

Метатрон 5.2

Метатрон

От Anastasiia Derzhanskaia

✓ Использование модели, рекомендованной создателем: GPT-5.2 Instant

GPT-ассистент для изучения основ программирования на языке Python

Вывести список тем

Спросите ChatGPT

Метатрон 5.2 Поделиться

Вывести список тем

Список тем курса (см.):

1. **Тема 1 — Вывод и ввод данных. Переменные.**
2. **Тема 2 — Арифметические операции в Python**
3. **Тема 3 — Условный оператор в Python**

Выберите тему, которую хотите изучить.

Спросите ChatGPT

Метатрон 5.2 Поделиться

Выберите тему, которую хотите изучить.

2

Раздел 1. Целые числа

1.1. Числовые данные в Python

В предыдущей теме мы работали в основном со строками. Это естественно, потому что команда `input()` по умолчанию считывает именно **текст**. Даже если пользователь введёт с клавиатуры число, Python сначала воспримет его как строку.

Однако в программировании очен то приходится работать не с текстом, а с **числами**. Например, программа может складывать значения, вычислять площадь фигуры, находить среднее арифметическое,

Спросите ChatGPT

Метатрон 5.2 Поделиться

В этой программе переменные `num1` и `num2` содержат целые числа, а переменная `num3` получает результат их сложения.

Теперь сравним это с другой программой:

Python Запустить

```
num1 = '7'
num2 = '10'
num3 = num1 + num2

print(num3)
```

Вывод программы:

```
710
```

На экране мы видим уже "склейку" этих чисел, потому что сложение двух строк склеивает их вместе.

Запомни:

Спросите ChatGPT

Метатрон 5.2 Поделиться

Можешь привести пару примеров, я не очень понял, почему 3 + 5 = 15

Если вы пишете:

Python Запустить

```
print(3 + 5)
```

то результат будет:

```
8
```

Потому что **3 и 5 — это числа**, записанные **без кавычек**, и выполняется обычное сложение.

А вот если написать так:

Python Запустить

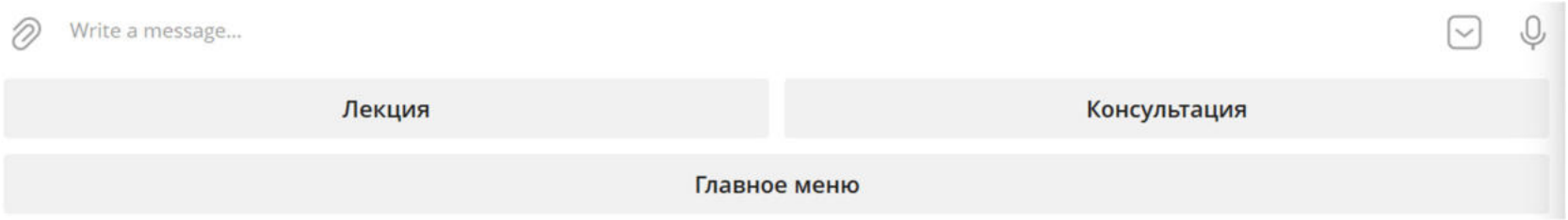
Спросите ChatGPT

Проблемы

- GPT-ассистент «слишком умный», чтобы просто воспроизводить текст лекции.
- Со сменой модели кардинально меняется поведение.
- Больше курс — дороже контент.

Решение

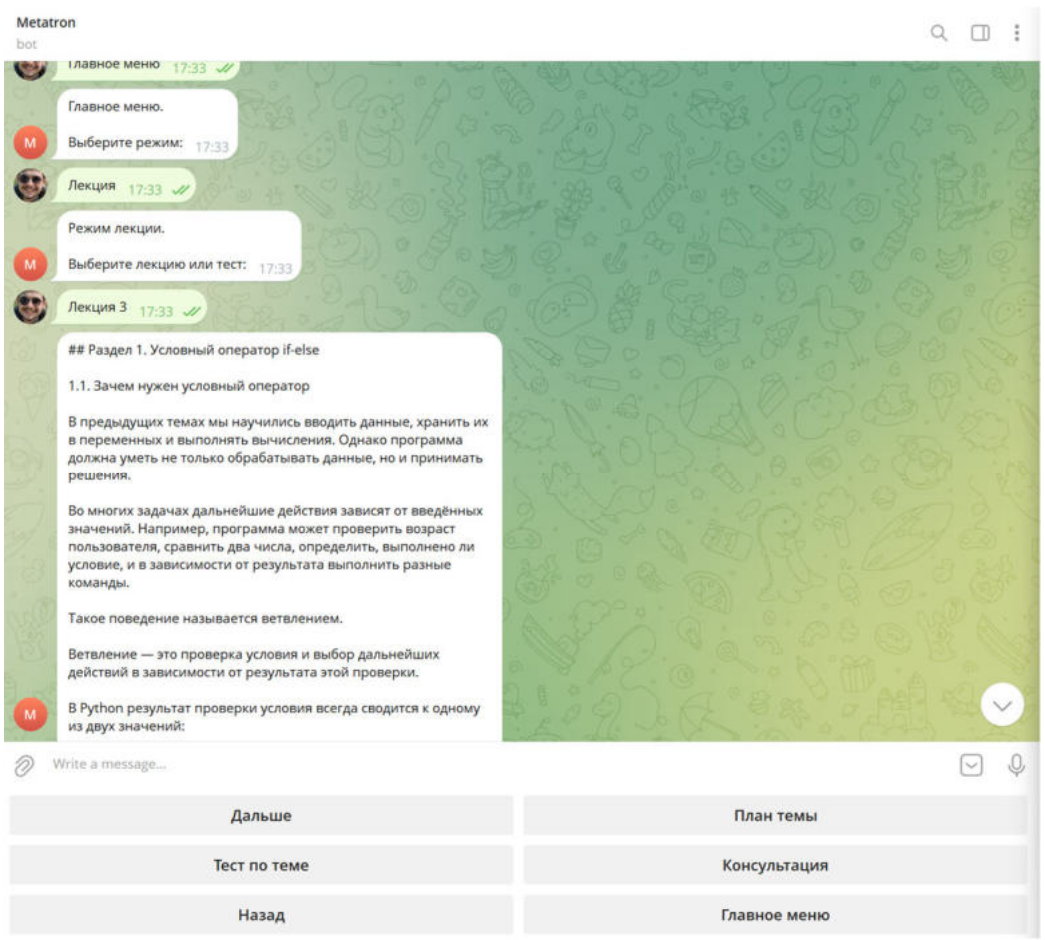
Режимы

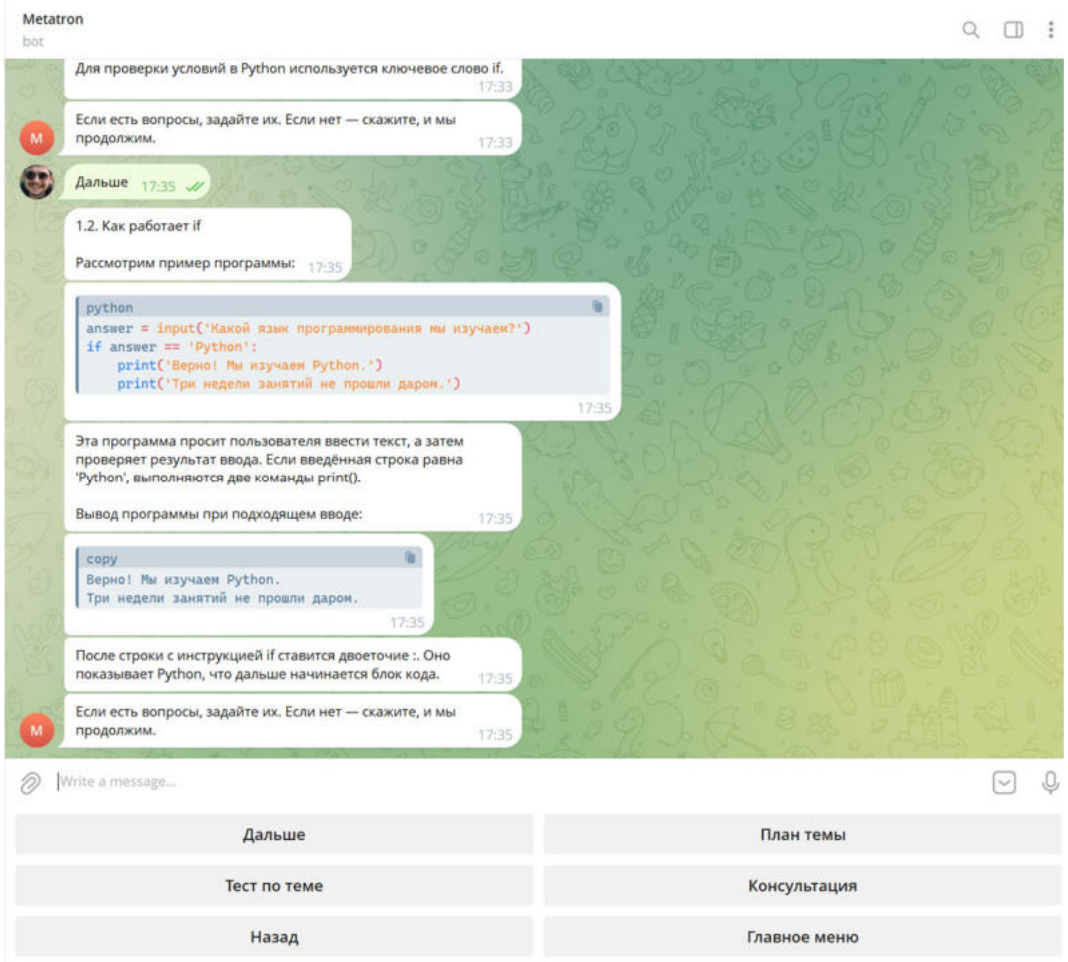


- Лекция:
  - Пошаговые объяснения темы.
  - Вопросы в рамках лекции.
- Тест:
  - Последовательные вопросы.
  - Изменение формулировки по запросу студента.
- Консультация:
  - Вопросы по ранее изученному материалу.
  - Ступенчатая помощь при решении задач.

ЮФУ ФАКУЛЬТЕТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК

Лекция

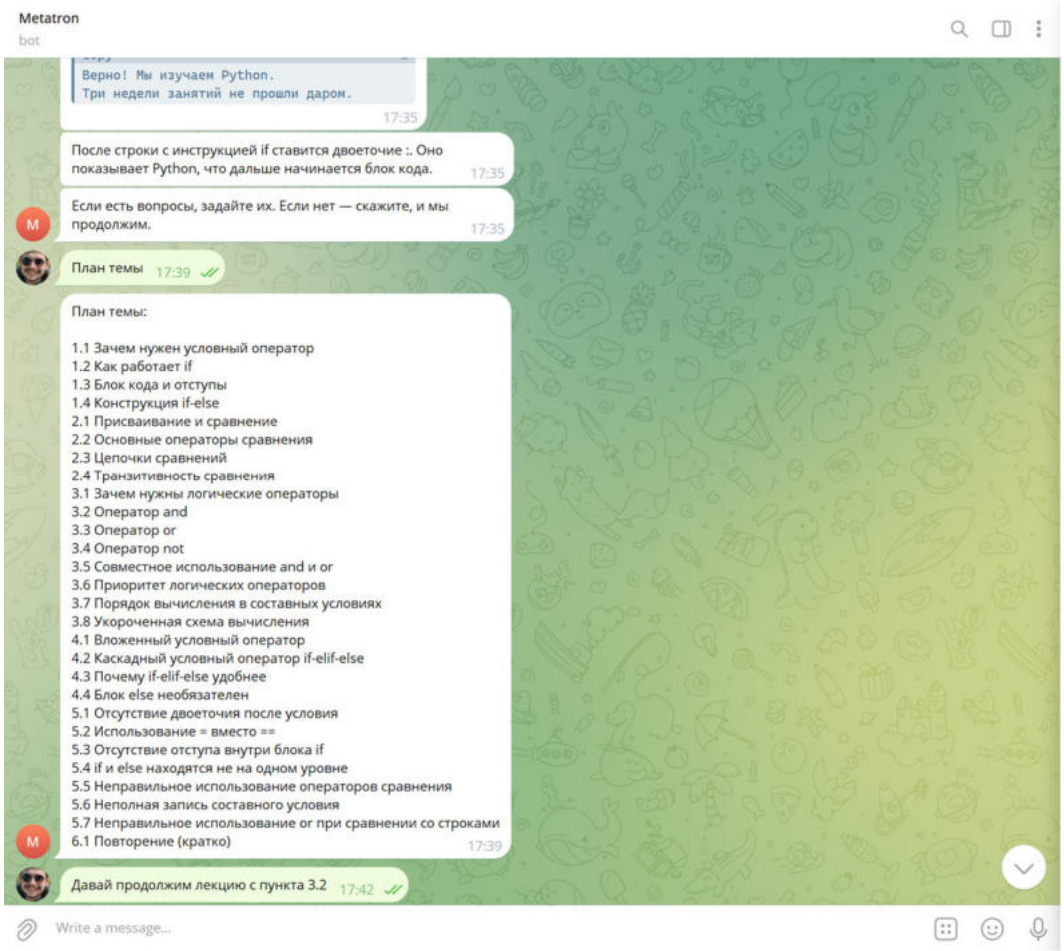


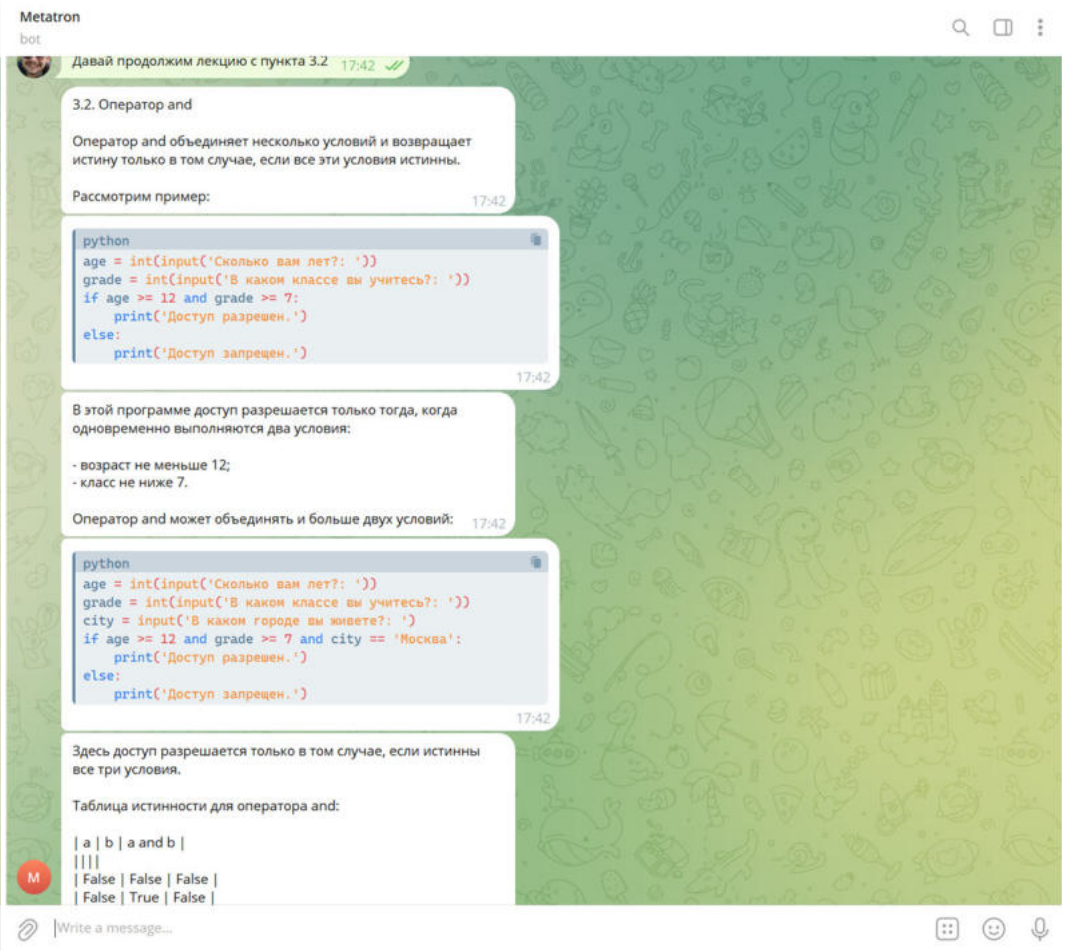


ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ ЮФУ

ФАКУЛЬТЕТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК

Лекция





ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ ЮФУ

ФАКУЛЬТЕТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК

Лекция

```
[TOPIC START: 3]
# Тема 3. Условный оператор в Python

[TOPIC MAP START: 3]
TOPIC_ID: TOPIC_3
TOPIC_TITLE: Тема 3. Условный оператор в Python

[SECTION 1]
1.1 -> BLOCK_3_1_1 -> Зачем нужен условный оператор
1.2 -> BLOCK_3_1_2 -> Как работает if
1.3 -> BLOCK_3_1_3 -> Блок кода и отступы
1.4 -> BLOCK_3_1_4 -> Конструкция if-else

[SECTION 2]
2.1 -> BLOCK_3_2_1 -> Присваивание и сравнение
2.2 -> BLOCK_3_2_2 -> Основные операторы сравнения
2.3 -> BLOCK_3_2_3 -> Цепочки сравнений
2.4 -> BLOCK_3_2_4 -> Транзитивность сравнения

[SECTION 3]
3.1 -> BLOCK_3_3_1 -> Зачем нужны логические операторы
3.2 -> BLOCK_3_3_2 -> Оператор and
3.3 -> BLOCK_3_3_3 -> Оператор or
3.4 -> BLOCK_3_3_4 -> Оператор not
3.5 -> BLOCK_3_3_5 -> Совместное использование and и or
3.6 -> BLOCK_3_3_6 -> Приоритет логических операторов
3.7 -> BLOCK_3_3_7 -> Порядок вычисления в составных условиях
3.8 -> BLOCK_3_3_8 -> Укороченная схема вычисления

[SECTION 4]
4.1 -> BLOCK_3_4_1 -> Вложенный условный оператор
4.2 -> BLOCK_3_4_2 -> Каскадный условный оператор if-elif-else
4.3 -> BLOCK_3_4_3 -> Почему if-elif-else удобнее
4.4 -> BLOCK_3_4_4 -> Блок else необязателен

[SECTION 5]
5.1 -> BLOCK_3_5_1 -> Отсутствие двоеточия после условия
5.2 -> BLOCK_3_5_2 -> Использование = вместо ==
5.3 -> BLOCK_3_5_3 -> Отсутствие отступа внутри блока if
5.4 -> BLOCK_3_5_4 -> if и else находятся не на одном уровне
5.5 -> BLOCK_3_5_5 -> Неправильное использование операторов сравнения
5.6 -> BLOCK_3_5_6 -> Неполная запись составного условия
5.7 -> BLOCK_3_5_7 -> Неправильное использование or при сравнении со строками

[SECTION 6]
6.1 -> BLOCK_3_6_1 -> Повторение (кратко)
[TOPIC MAP END: 3]
```

```
[BLOCK START: BLOCK_3_1_3]
[BLOCK NUMBER: 1.3]
[BLOCK TITLE: Блок кода и отступы]
### 1.3. Блок кода и отступы

**Блок кода** — это несколько строк программы, которые относятся к одной инструкции и выполняются как единое целое.

В Python блоки кода выделяются с помощью **отступов**. Это не просто вопрос аккуратного оформления. Для Python отступы являются частью синтаксиса, поэтому без них программа либо не запустится, либо будет работать неправильно.

В других языках программирования границы блока могут обозначаться специальными символами, например фигурными скобками. В Python для этого используются именно отступы. По ним интерпретатор определяет, какие строки относятся к условному оператору, а какие уже находятся за его пределами.

**Отступ** — это небольшое смещение строки вправо. Обычно в начале такой строки стоят пробелы.

Схема записи условного оператора выглядит так:

```python
if условие:
    блок кода
```

Все строки с отступом после if считаются частью одного блока. Если условие истинно, выполняется весь этот блок. Как только Python встречает строку без нужного отступа, он считает, что блок закончился. Если условие ложно, весь блок после if пропускается.

Поэтому отступы в Python критически важны: они не просто делают код более читаемым, но напрямую влияют на работу программы.

По соглашению PEP 8 для одного уровня отступа используют 4 пробела.

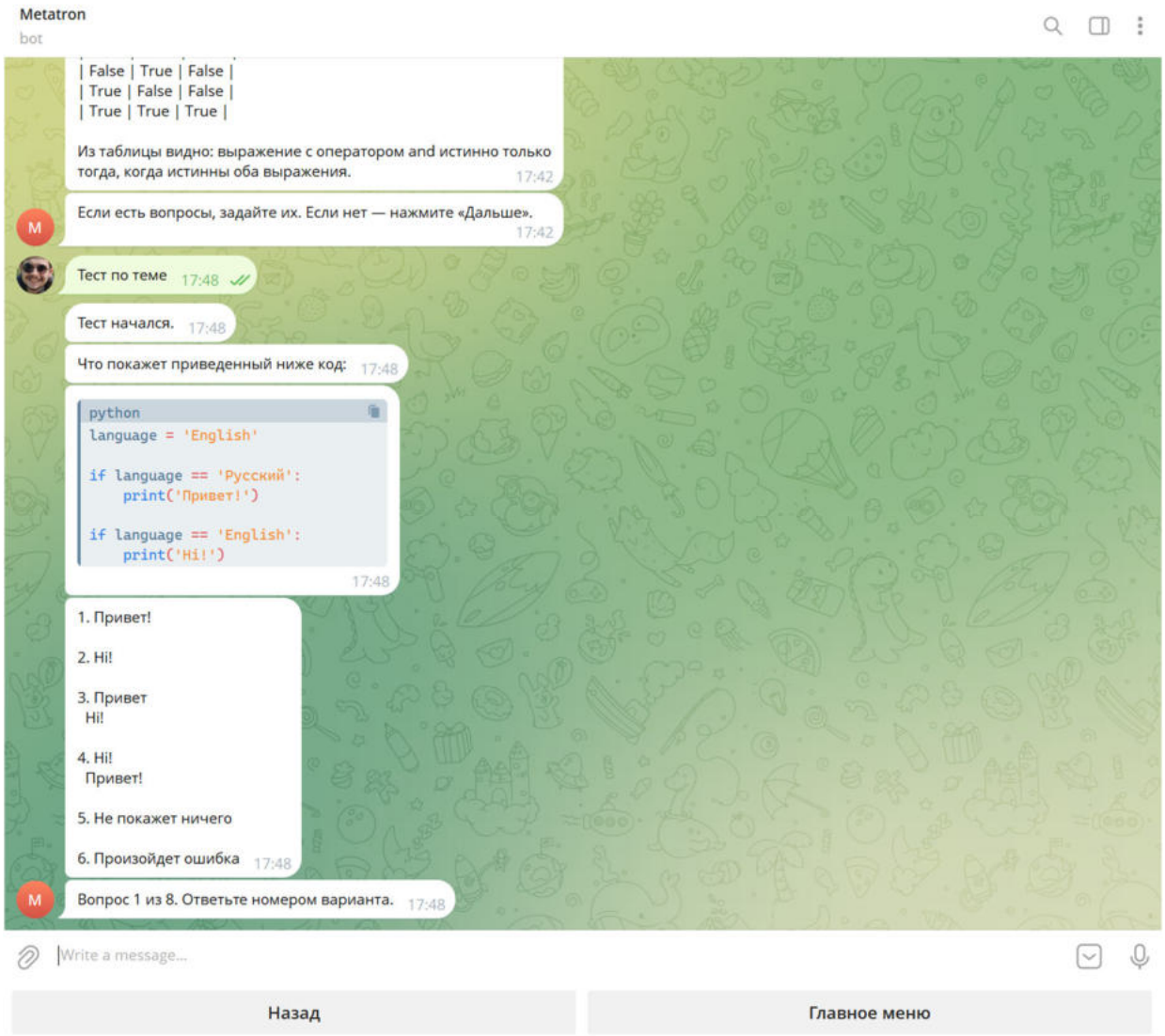
[BLOCK END: BLOCK_3_1_3]

[BLOCK START: BLOCK_3_1_4]
[BLOCK NUMBER: 1.4]
```

ЮФУ

ФАКУЛЬТЕТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК

Тест



ЮФУ

ФАКУЛЬТЕТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК

Консультация

Элементы контроля:

- Промпт.
- Постобработка ответа модели.

Консультация

Промпт:

- Перечислены пройденные темы и запрещены все остальные.
- Зафиксирован принцип минимально достаточной помощи и задано поэтапоное усиление поддержки.
- Полное решение запрещено.

Консультация

Этапы проверки при постобработке:

- Выход за рамки пройденного материала:
  - Маркеры непройденных тем.
- Слишком сильная помощь.
- Не является ли ответ уже готовым решением:
  - По фразам.
  - По объему fenced-блоков.

ЮФУ ФАКУЛЬТЕТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК

Безопасность системы

- Валидация контента при запуске.
- Разделение состояния пользователя.
- Whitelist пользователей.

Вывод

- В предметно ограниченных курсах технически возможно построение LLM-ассистентов, которые устойчиво работают в пределах хронологии и содержания курса.
- Полученное решение остается модульным и переносимым, что позволяет адаптировать его к другими платформам и курсам.

ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ЮФУ
ФАКУЛЬТЕТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК