



**Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)**

**ФАКУЛЬТЕТ**

**ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

**КАФЕДРА**

**СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ**

**Отчет по лабораторной работе №6  
по дисциплине Базовые компоненты интернет технологии**

**Тема работы: "Разработка бота на основе конечного автомата для  
Telegram с использованием языка Python"**

**Выполнил:**

Студент группы ИУ5Ц-54Б

Перевощиков Н.Д.

09.12.22 г.

(дата, подпись)

**Проверил:**

Преподаватель

Канев А.И.

(дата, подпись)

Москва, 2022

---

## СОДЕРЖАНИЕ ОТЧЕТА

|                                      |    |
|--------------------------------------|----|
| 1. Цель лабораторной работы.....     | 3  |
| 2. Описание задания.....             | 3  |
| 3. Текст программы.....              | 4  |
| __init__.py .....                    | 4  |
| Config.py .....                      | 4  |
| Arithmetic_calculate.py.....         | 4  |
| Bot_telegram.py .....                | 6  |
| Json_function.py .....               | 9  |
| Work_calculate.py.....               | 10 |
| NFS.jpg .....                        | 11 |
| 4. Результаты работы программы ..... | 12 |

## **1. Цель лабораторной работы**

Изучение разработки ботов в Telegram.

## **2. Описание задания**

Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4.

1. Разработайте бота для Telegram. Бот должен реализовывать конечный автомат из трех состояний.

### 3. Текст программы

#### Config.py

Здесь должен быть токен для доступа к HTTP API

```
token = ''
```

#### \_\_init\_\_.py

```
print('Directory calculate')
```

#### Arithmetic\_calculate.py

```
class mathematical_calculator(object):
    def __init__(self, math_calculate):
        self.math_calculation = math_calculate
        self.math_calculation_list =
self.convert_string_in_list(math_calculate)
        self.list_enumeration_of_sign =
self.enumeration_of_sign(self.math_calculation_list)

        self.type_error = None

        for sign in self.list_enumeration_of_sign:
            self.arithmetic_operations(sign, self.math_calculation_list)

        if(self.type_error == None):
            self.result = float(self.math_calculation_list[0])

# Преобразование тип строки в list
def convert_string_in_list(self, str_calculate):
    str_1 = []
    str_meaning = ''
    for i in str_calculate:
        if(i != ' '):
            str_meaning += i
        else:
            str_1.append(str_meaning)
            str_meaning = ''

    #append () добавляет в конец списка элемент, переданный ему в
    #качестве аргумента.
    str_1.append(str_meaning)

    return str_1

#Растановка приоритета операции
def enumeration_of_sign(self, list_str):
    count_list = []
    for i in list_str:
        if ('*' == i): count_list.append(i)
        if ('/' == i): count_list.append(i)
        if ('+' == i): count_list.append(i)
        if ('-' == i): count_list.append(i)

    count_list = self.priority(count_list)

    return count_list

#Поддержка функции по расстановку приоритета операции
def priority(self, list_str):
    list_1 = []
    size = len(list_str)
    count = 0
    while (size != 0):
        if ('*' in list_str or '/' in list_str):
```

```

        for i in list_str:
            if (i == '*' or i == '/'):
                list_1.append(i)
            size -= 1
        if ('+' in list_str or '-' in list_str):
            for i in list_str:
                if (i == '+' or i == '-'):
                    list_1.append(i)
                size -= 1

    return list_1

#Арифметические операции
def arithmetic_operations(self, sign, list):
    result = None
    if (sign in list):
        for i in range(1, len(list)-1):
            try:
                if (list[i] == sign):
                    if (sign == '*'):
                        result = float(list[i - 1]) * float(list[i +
1]))

                    elif (sign == '/'):
                        result = float(list[i - 1]) / float(list[i +
1]))

                    elif (sign == '+'):
                        result = float(list[i - 1]) + float(list[i +
1]))

                    elif (sign == '-'):
                        result = float(list[i - 1]) - float(list[i +
1]))

                list[i] = result
                del list[i - 1: i]
                del list[i: i + 1]

            #Деление на 0
            except ZeroDivisionError:
                self.type_error = 'Division by 0'
                self.result = 'infinity'

            #Граница вне диапазона
            except:
                return result

    def calculate(self, math_calculate):
        self.math_calculation = math_calculate
        self.math_calculation_list =
self.convert_string_in_list(math_calculate)
        self.list_enumeration_of_sign =
self.enumeration_of_sign(self.math_calculation_list)

        self.type_erorr = None

        for sign in self.list_enumeration_of_sign:
            self.arithmetic_operations(sign, self.math_calculation_list)

        if(self.type_erorr == None):
            self.result = float(self.math_calculation_list[0])

    return self

```

## Bot\_telegram.py

```
import config
import telebot
from telebot import types
import random

#SageMathCloud (сокращённо SMC) — это онлайн-сервис, в котором можно
написать математический или любой другой расчёт.
from arithmetic_calculate import mathematical_calculator as smc
from json_function import combined_data, delete_data_for_id_user,
load_data_for_id_user
from work_calculate import generator_of_meaning

# Создание бота
bot = telebot.TeleBot(config.token)

SPISOK = '''
/menu - Меню (кнопки переключателя)
/calculate - Бот-калькулятор (посчет арифметических операций)
/story - Просмотр история вычисления
/cleaner - Очистка истории вычисления
/generation - Генерация случайных вычислений
/image - Просмотр изображения
'''

# Список меню
@bot.message_handler(commands=['spisok'])
def menu(message):
    bot.send_message(message.chat.id, SPISOK)

# При нажатии на /menu
@bot.message_handler(commands=['menu'])
def menu(message):
    markup = types.InlineKeyboardMarkup(row_width=1)
    btn_1 = types.InlineKeyboardButton(text="Решить пример",
callback_data='btn_1')
    btn_2 = types.InlineKeyboardButton(text="Посмотреть историю
вычисления", callback_data='btn_2')
    btn_3 = types.InlineKeyboardButton(text="Очистить истории вычисления",
callback_data='btn_3')
    btn_4 = types.InlineKeyboardButton(text="Генерировать вычисления",
callback_data='btn_4')
    btn_5 = types.InlineKeyboardButton(text="Посмотреть изображение",
callback_data='btn_5')
    markup.add(btn_1, btn_2, btn_3, btn_4, btn_5)
    bot.send_message(message.chat.id, text=f"Привет,
{message.from_user.first_name}! Выберите то, что Вам нужно",
reply_markup=markup)

# Кнопки переключателя при нажатии на /menu
@bot.callback_query_handler(func=lambda callback: callback.data)
def check_callback_data(callback):
    #Пользовательский индентификатор
    user_id = str(callback.from_user.id)

    if (callback.data == "btn_1"):
        bot.send_message(callback.message.chat.id, 'Напишите пример
вычисления')

    # Пользовательский индентификатор
    user_id = str(callback.from_user.id)

    @bot.message_handler(content_types=["text"])
```

```

def info(message):
    meaning = smc(message.text)
    bot.send_message(message.chat.id, f'Решение:
{meaning.result}')
    data = {
        user_id: [{"id": random.randint(0, 10000),
                    "meaning": str(message.text),
                    "result": str(meaning.result)}]
    }
    combined_data(data, str(message.from_user.id))

elif (callback.data == "btn_2"):
    bot.send_message(callback.message.chat.id, 'История вычисления')
    data = load_data_for_id_user(str(user_id))
    if (data == 'Ошибка! Такого идентификатора не существует.'):
        bot.send_message(callback.message.chat.id, 'Нет базы данных')
    else:
        for j in range(len(data) - 1):
            id = data[j]['id']
            meaning = data[j]['meaning']
            result = data[j]['result']
            print_info = f'id:{id}\n{meaning} = {result}\n\n'
            bot.send_message(callback.message.chat.id, print_info)

elif (callback.data == "btn_3"):
    bot.send_message(callback.message.chat.id, 'Очистка истории
вычисления')

    check_error = delete_data_for_id_user(user_id)

    if (check_error != 'Ошибка! Такого идентификатора не существует.'):
        bot.send_message(callback.message.chat.id, 'Операция прошла
успешно')
    else:
        bot.send_message(callback.message.chat.id, check_error)

elif (callback.data == "btn_4"):
    bot.send_message(callback.message.chat.id, 'Генерация случайных
вычислений')
    generator_of_meaning(user_id)
    bot.send_message(callback.message.chat.id, 'Операция прошла
успешно')

elif (callback.data == "btn_5"):
    img = open('NFS.jpg', 'rb')
    bot.send_photo(callback.message.chat.id, img)

else:
    bot.send_message(callback.chat.id, 'Нет такой команды. Введите
/spisok')

# Вычисления
@bot.message_handler(commands=['calculate'])
def start_calculate(message):
    bot.send_message(message.chat.id, 'Напишите пример вычисления')

    # Пользовательский идентификатор
    user_id = str(message.from_user.id)

    @bot.message_handler(content_types=["text"])
    def info(message):
        meaning = smc(message.text)
        bot.send_message(message.chat.id, f'Результат решения:
{meaning.result}')

```

```

        data = {
            user_id: [{"id": random.randint(0, 10000),
                        "meaning": str(message.text),
                        "result": str(meaning.result)}]
        }
        combined_data(data, str(message.from_user.id))

# Просмотр история вычисления
@bot.message_handler(commands=['story'])
def start_story(message):
    bot.send_message(message.chat.id, 'История вычисления')

    # Пользовательский идентификатор
    user_id = str(message.from_user.id)

    data = load_data_for_id_user(str(user_id))

    if (data == 'Ошибка! Такого идентификатора не существует.'):
        bot.send_message(message.chat.id, 'Нет базы данных')
    else:
        for j in range(len(data) - 1):
            id = data[j]['id']
            meaning = data[j]['meaning']
            result = data[j]['result']
            print_info = f'id: {id}\n{meaning} = {result}\n\n'
            bot.send_message(message.chat.id, print_info)

#Просмотр изображения
@bot.message_handler(commands=['image'])
def start_image(message):
    img = open('NFS.jpg', 'rb')
    bot.send_photo(message.chat.id, img)

#Генерация случайных вычислений
@bot.message_handler(commands=['generation'])
def start_generation(message):
    bot.send_message(message.chat.id, 'Генерация случайных вычислений')

    #Пользовательский идентификатор
    user_id = str(message.from_user.id)

    generator_of_meaning(user_id)

    bot.send_message(message.chat.id, 'Операция прошла успешно')

#Очистка истории вычисления
@bot.message_handler(commands=['cleaner'])
def start_cleaner(message):
    bot.send_message(message.chat.id, 'Очистка истории вычислений')

    # Пользовательский идентификатор
    user_id = str(message.from_user.id)

    check_error = delete_data_for_id_user(user_id)

    if (check_error != 'Ошибка! Такого идентификатора не существует.'):
        bot.send_message(message.chat.id, 'Операция прошла успешно')
    else:
        bot.send_message(message.chat.id, check_error)

#Работа программы в телеграме без остановки
bot.polling(none_stop=True)

```



## Json\_function.py

```
import json

file_locator = 'D:\Работа\МГТУ им.  
Н.Э.Баумана\Программирование\Программы\Программы за 5  
семестр\LAB_06\calculate\data'

#Запись данных
def data_recording(data, title=file_locator):
    with open(f"{title}.json", "w", encoding="utf-8") as file:
        json.dump(data, file, indent=2, ensure_ascii=False)

#Чтение данных
def load_data(title=file_locator):
    with open(f"{title}.json", "r") as file:
        data = json.load(file)
    return data

#Добавление данных
def combined_data(data_json, id_user='id_user', title=file_locator):
    #Если есть файл и не пустой
    try:
        with open(f"{title}.json", encoding="utf-8") as file:
            data = json.load(file)
            temp = data[id_user]
            for info_data in data_json[id_user]:
                n = {
                    'id': info_data['id'],
                    'meaning': info_data['meaning'],
                    'result': info_data['result']
                }
            temp.append(n)
        data_recording(data)
    #Если нет файла
    except:
        data_recording(data_json)

#Загрузка данных для идентификатора пользователя
def load_data_for_id_user(id_user, title=file_locator):
    try:
        with open(f"{title}.json", "r", encoding="utf-8") as file:
            data = json.load(file)
            temp = data[id_user]
            for info_data in data[id_user]:
                n = {
                    'id': info_data['id'],
                    'meaning': info_data['meaning'],
                    'result': info_data['result']
                }
            temp.append(n)
        return temp
    except:
        return 'Ошибка! Такого идентификатора не существует.'

#Удаление данных для индентификатора пользователя
def delete_data_for_id_user(id_user, title=file_locator):
    try:
        with open(f"{title}.json", encoding="utf-8") as file:
            data = json.load(file)
            data_1 = {}
            for id_user_data in data:
                if (id_user != id_user_data):
                    temp = data[id_user_data]
```

```

        data_1 = {id_user_data: []}
        for j in temp:
            n = {
                'id': j['id'],
                'meaning': j['meaning'],
                'result': j['result']
            }
            data_1[id_user_data].append(n)
        temp.append(data_1)
    data_recording(data_1)
except:
    return 'Ошибка! Такого идентификатора не существует.'

```

## Work\_calculate.py

```

import random

from calculate.json_function import data_recording, load_data, combined_data,
load_data_for_id_user
from calculate.arithmetic_calculate import mathematical_calculator as smc

#Генерация значений
def generator_of_meaning(id_user='id_user'):
    arithmetic = ['+', '-', '/', '*']

    arith = arithmetic[random.randint(0, 3)]
    gen_id = random.randint(0, 100000)
    m_1 = random.randint(0, 1000)
    m_2 = random.randint(0, 1000)
    class_calculate = smc(str(m_1) + ' ' + str(arith) + ' ' + str(m_2))

    data = {
        str(id_user): [
            {"id": gen_id,
             "meaning": (str(m_1) + ' ' + str(arith) + ' ' + str(m_2)),
             "result": class_calculate.result}
        ]
    }
    combined_data(data, id_user)

#Получение информации
def get_info():
    try:
        data = load_data()
        return data
    except:
        return 'Нет файла'

#Получение информации с идентификатором пользователя
def get_info_id_user(id_user):
    try:
        data = load_data_for_id_user(id_user)
        return data
    except:
        return 'Нет файла'

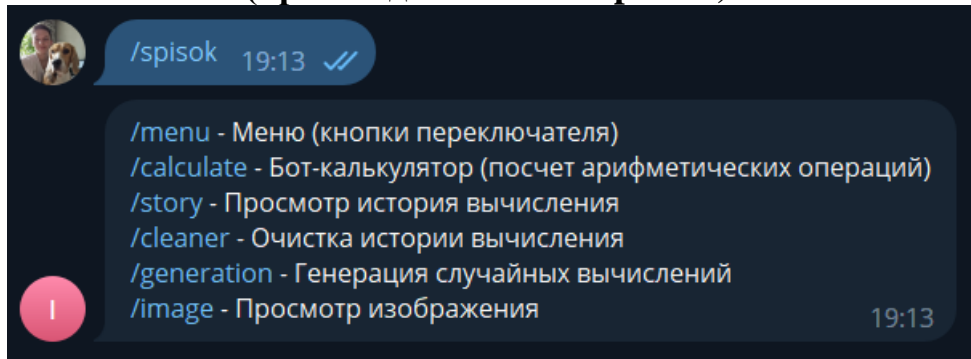
```

NFS.jpg

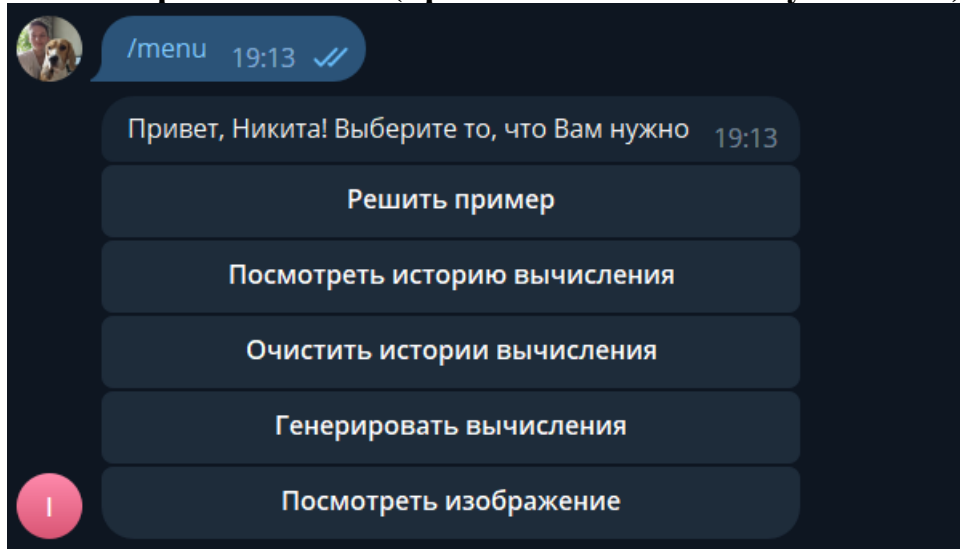


#### 4. Результаты работы программы

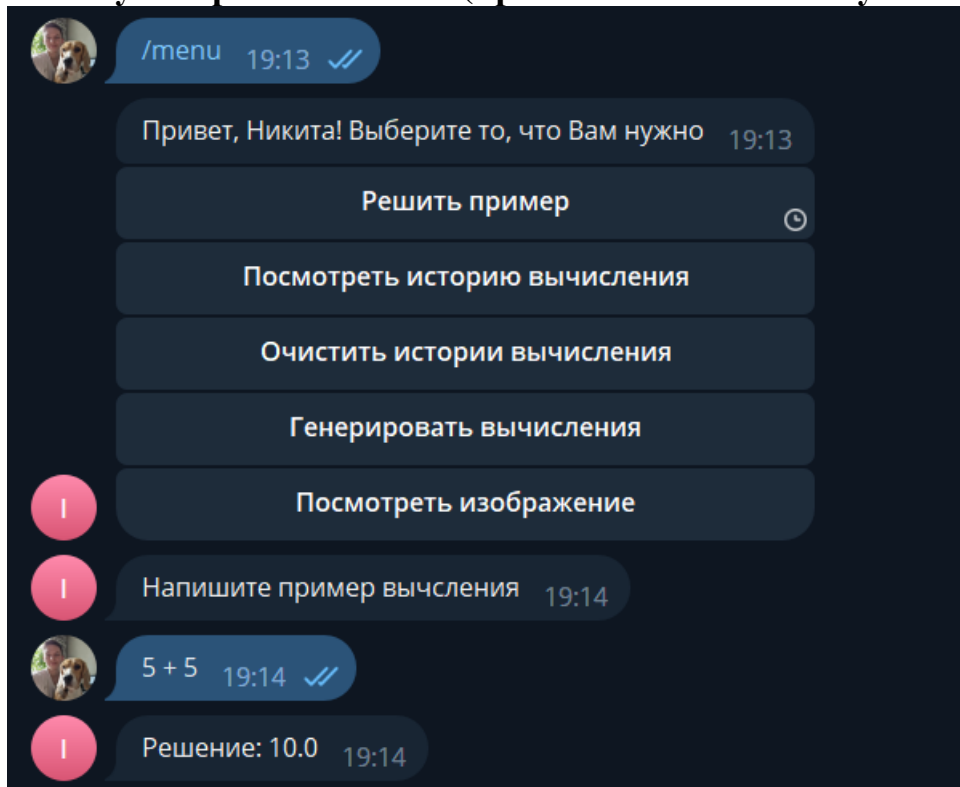
##### Список меню (при вводе ссылки «/spisok»)



##### Меню переключателя (при нажатии на ссылку «/menu»)

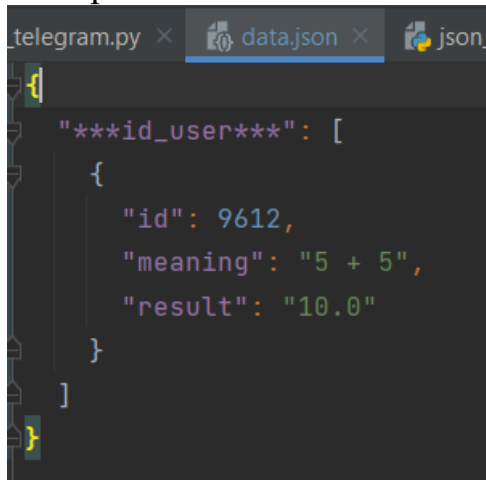


##### Калькулятор вычисления (при нажатии на кнопку «Решить пример»)



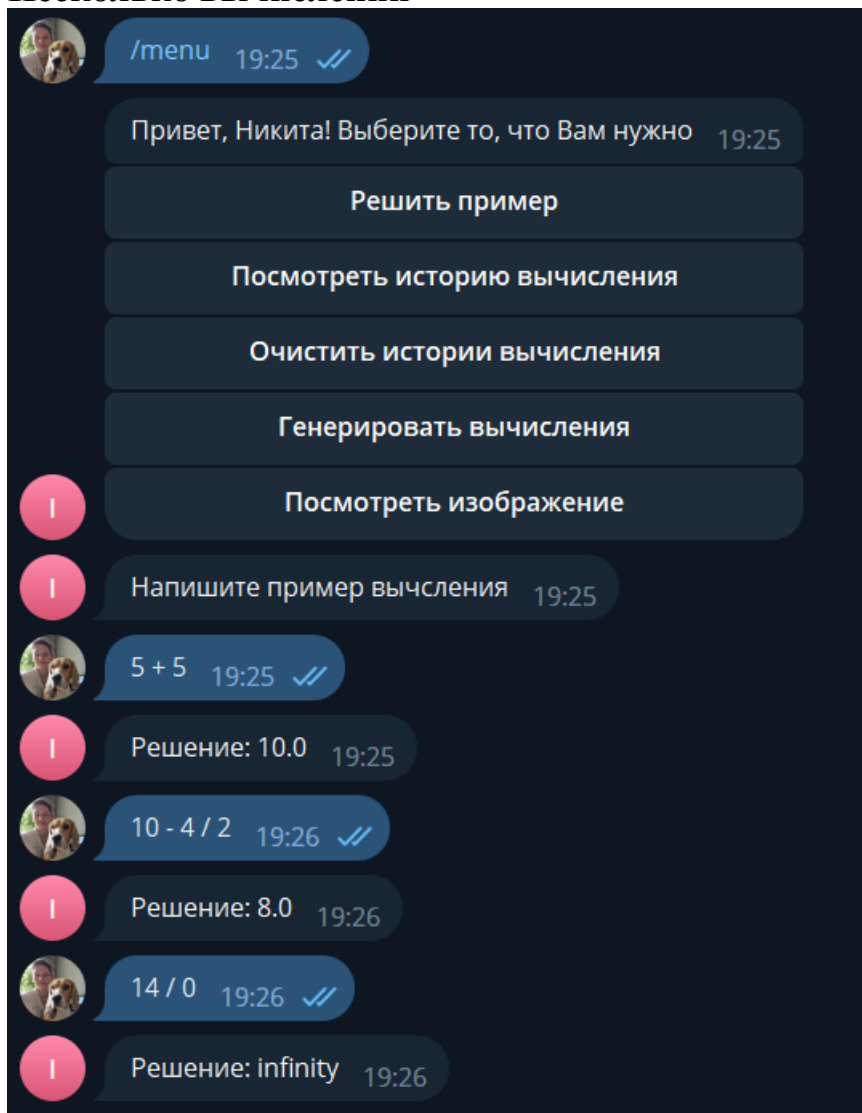
## Данные хранятся в data.json

Вместо "\*\*\*id\_user\*\*\*" должны быть цифры индекса пользователя из телеграмма.

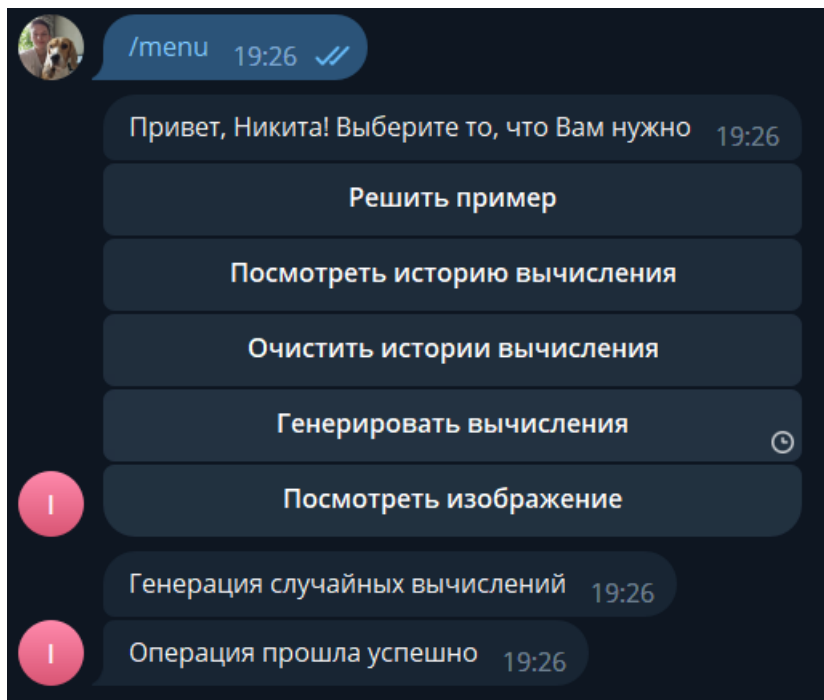


```
{
  "***id_user***": [
    {
      "id": 9612,
      "meaning": "5 + 5",
      "result": "10.0"
    }
  ]
}
```

## Несколько вычислений



Генерация случайных вычислений (при нажатии на кнопку «Генерировать вычисления»)

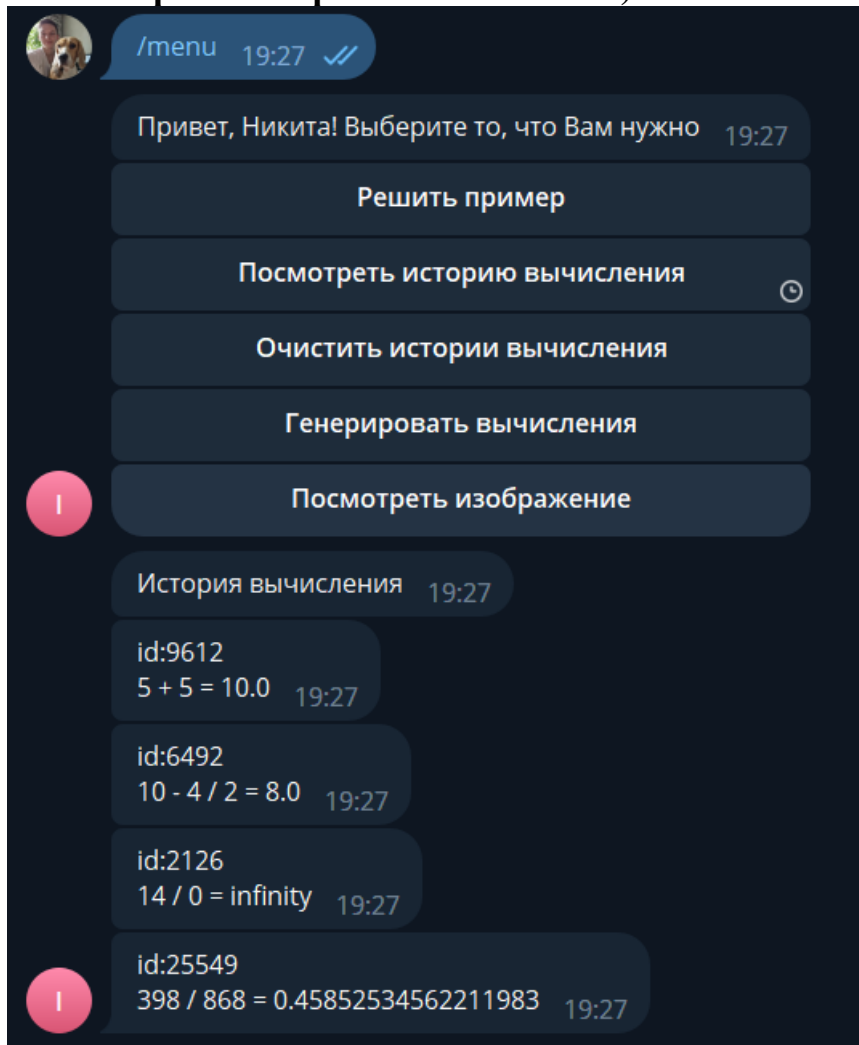


## Обновленные данные хранятся в data.json

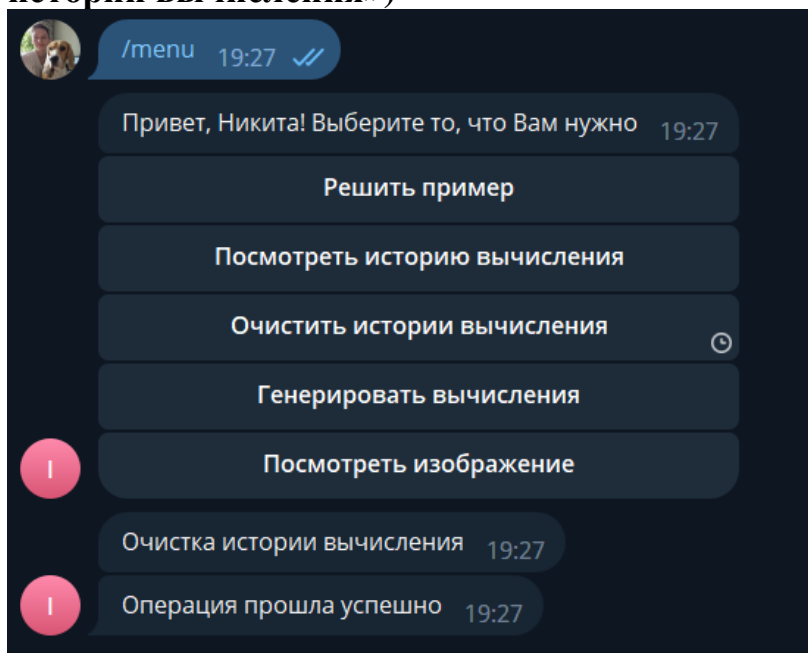
Вместо "\*\*\*id\_user\*\*\*" должны быть цифры индекса пользователя из телеграмма.

```
telegram.py × data.json × json_function.py
[
  {
    "***id_user***": [
      {
        "id": 9612,
        "meaning": "5 + 5",
        "result": "10.0"
      },
      {
        "id": 6492,
        "meaning": "10 - 4 / 2",
        "result": "8.0"
      },
      {
        "id": 2126,
        "meaning": "14 / 0",
        "result": "infinity"
      },
      {
        "id": 25549,
        "meaning": "398 / 868",
        "result": 0.45852534562211983
      }
    ]
  }
]
```

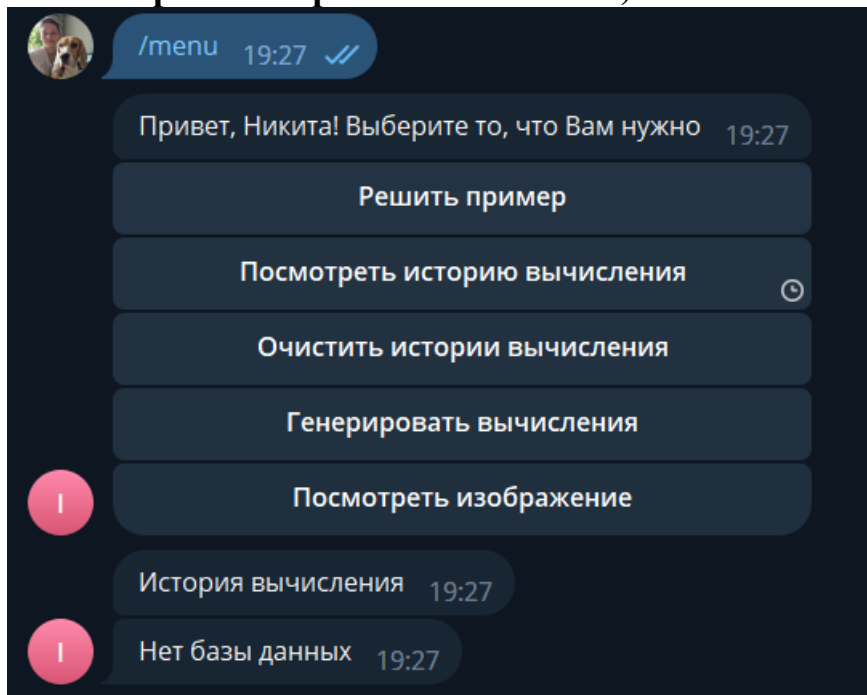
Чтение и просмотр данные из data.json (при нажатии на кнопку «Посмотреть историю вычисления»)



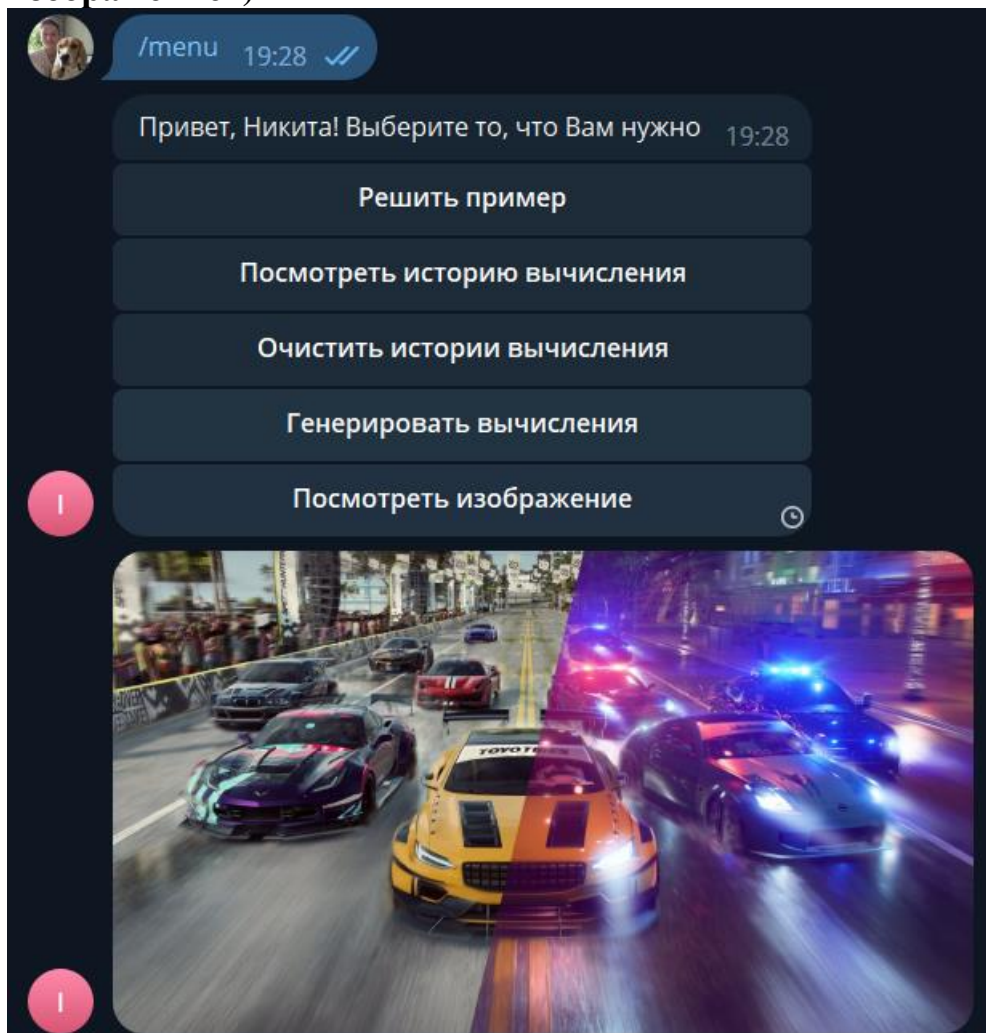
Очистка истории вычисления (при нажатии на кнопку «Очистить истории вычисления»)



Проверка, очистила ли история вычисления (при нажатии на кнопку «Посмотреть историю вычисления»)

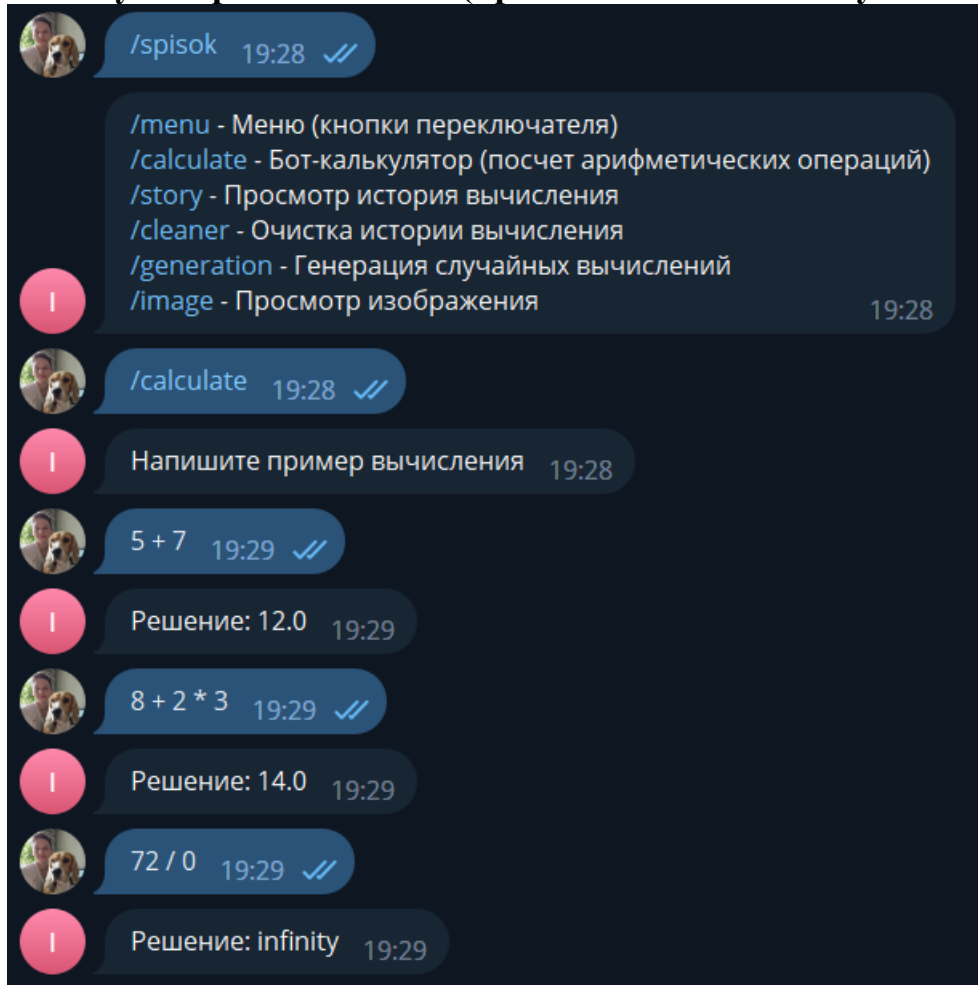


Просмотр изображения (при нажатии на кнопку «Посмотреть изображение»)

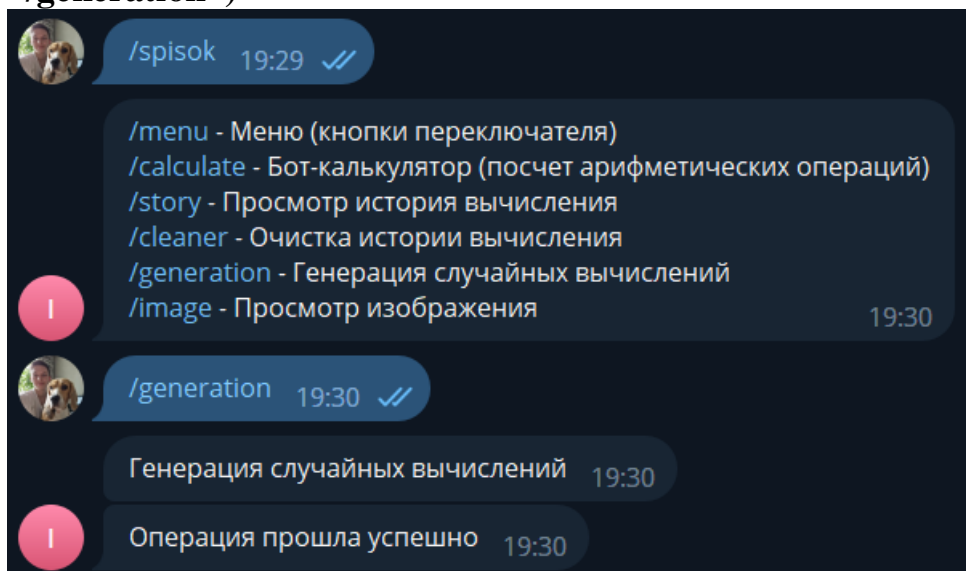




### Калькулятор вычисления (при нажатии на ссылку «/calculate»)



### Генерация случайных вычислений (при нажатии на ссылку «/generation»)

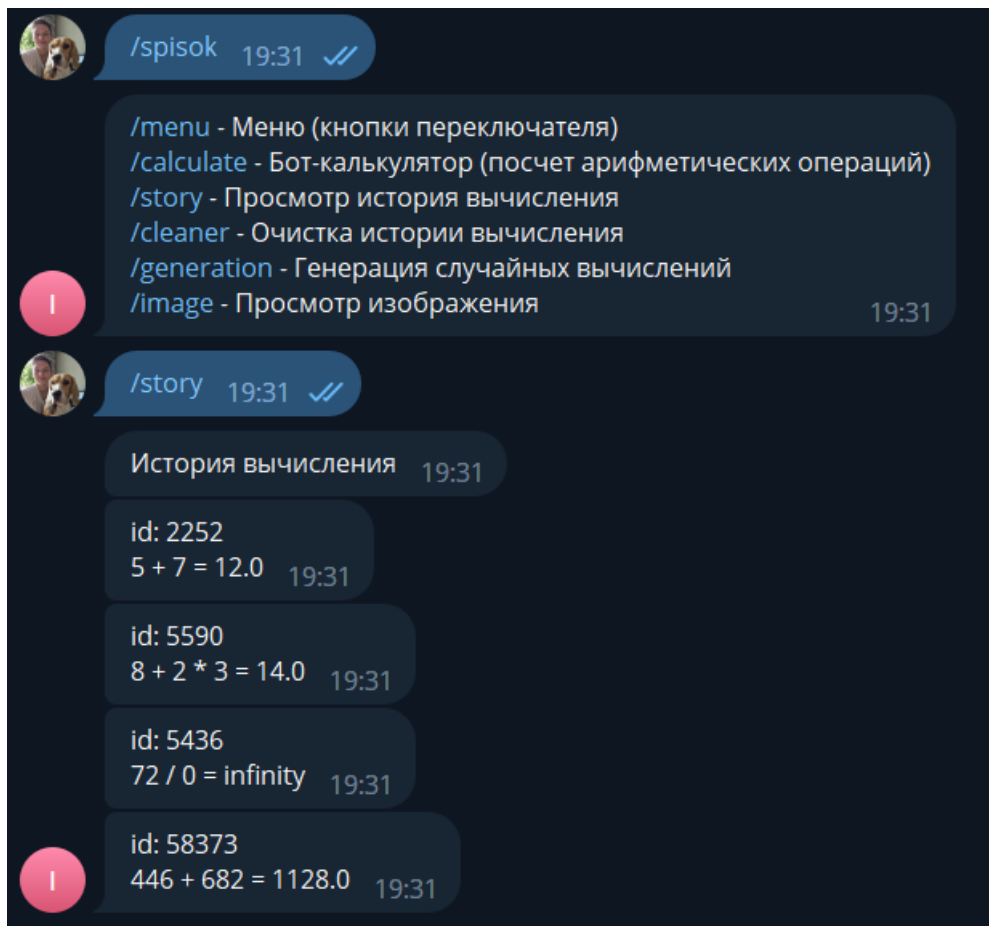


### Данные хранятся в data.json

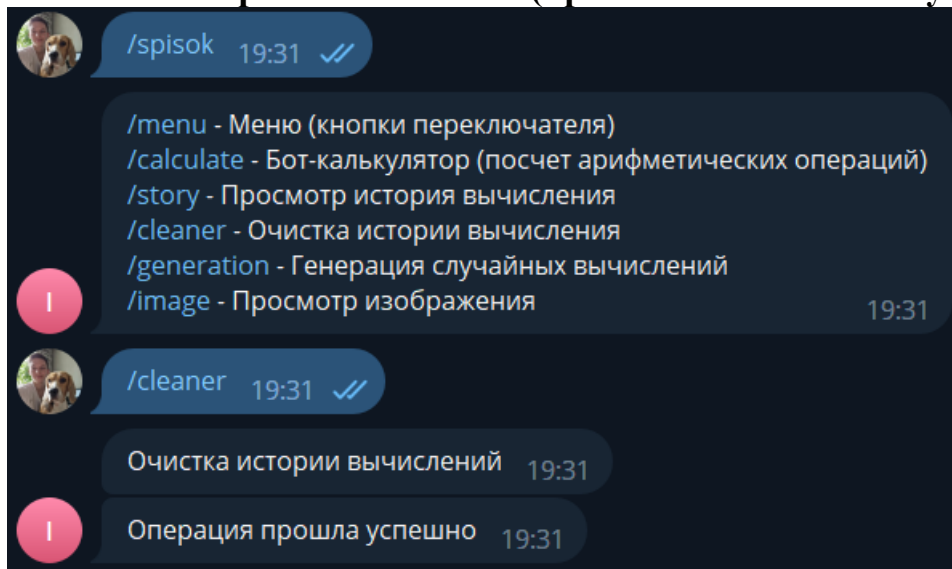
Вместо "\*\*\*id\_user\*\*\*" должны быть цифры индекса пользователя из телеграмма.

```
telegram.py × data.json × json_funcio
{
  "***id_user***": [
    {
      "id": 2252,
      "meaning": "5 + 7",
      "result": "12.0"
    },
    {
      "id": 5590,
      "meaning": "8 + 2 * 3",
      "result": "14.0"
    },
    {
      "id": 5436,
      "meaning": "72 / 0",
      "result": "infinity"
    },
    {
      "id": 58373,
      "meaning": "446 + 682",
      "result": 1128.0
    }
  ]
}
```

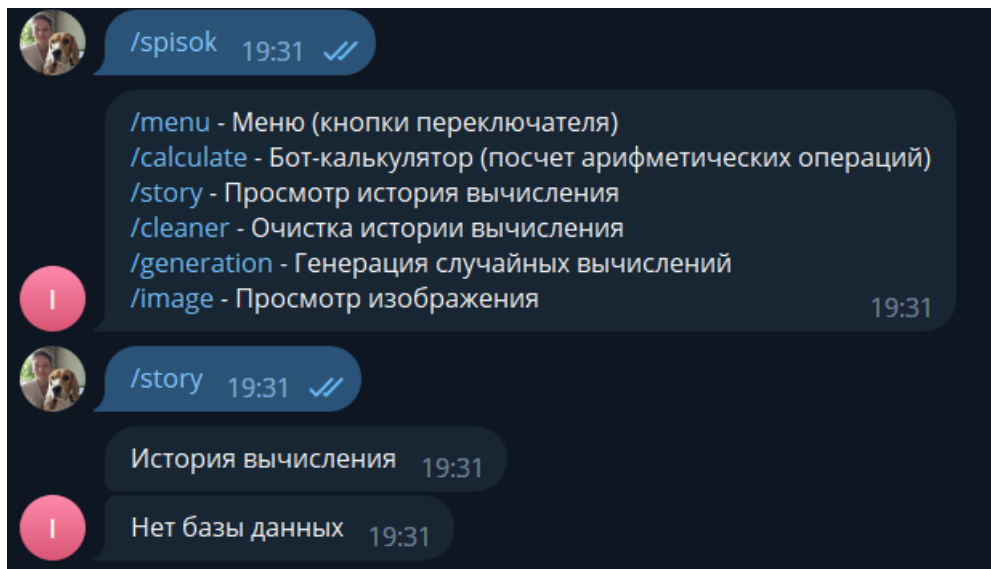
**Чтение и просмотр данные из data.json (при нажатии на ссылку «story»)**



### Очистка истории вычисления (при нажатии на ссылку «/cleaner»)



### Проверка, очистила ли история вычисления (при нажатии на ссылку «/story»)



### Просмотр фотографии (при нажатии на ссылку «/image»)

