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 домашней работе
по дисциплине Базовые компоненты интернет технологии

Выполнил:
Студент группы ИУ5Ц-54Б
Перевощиков Н.Д. 17.12.22 г.
(дата, подпись)

Проверил:
Преподаватель
Канев А.И.
(дата, подпись)

Москва, 2022

СОДЕРЖАНИЕ ОТЧЕТА

1. Цель домашнего задания

Изучение возможностей создания ботов в Telegram и их тестирования.

2. Описание задания

1. Модифицируйте код лабораторной работы №5 или №6 таким образом, чтобы он был пригоден для модульного тестирования.
2. Используя материалы лабораторной работы №4 создайте модульные тесты с применением TDD - фреймворка (2 теста) и BDD - фреймворка (2 теста).

3. Текст программы

Находится в папке «Calculate»:

3.1.config.py

Здесь должен быть токен для доступа к HTTP API

```
token = ''
```

3.2.__init__.py

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

3.3.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):
        counter_sign = []
        for i in list_str:
            if ('*' == i): counter_sign.append(i)
            if ('/' == i): counter_sign.append(i)
            if ('+' == i): counter_sign.append(i)
            if ('-' == i): counter_sign.append(i)

        counter_sign = self.priority(counter_sign)

        return counter_sign

    #Подержка 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
```

3.4.Bot_telegram.py

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

#SageMathCloud (сокращённо SMC) — это онлайновый 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('moscow.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('moscow.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)
```

3.5.Json_function.py

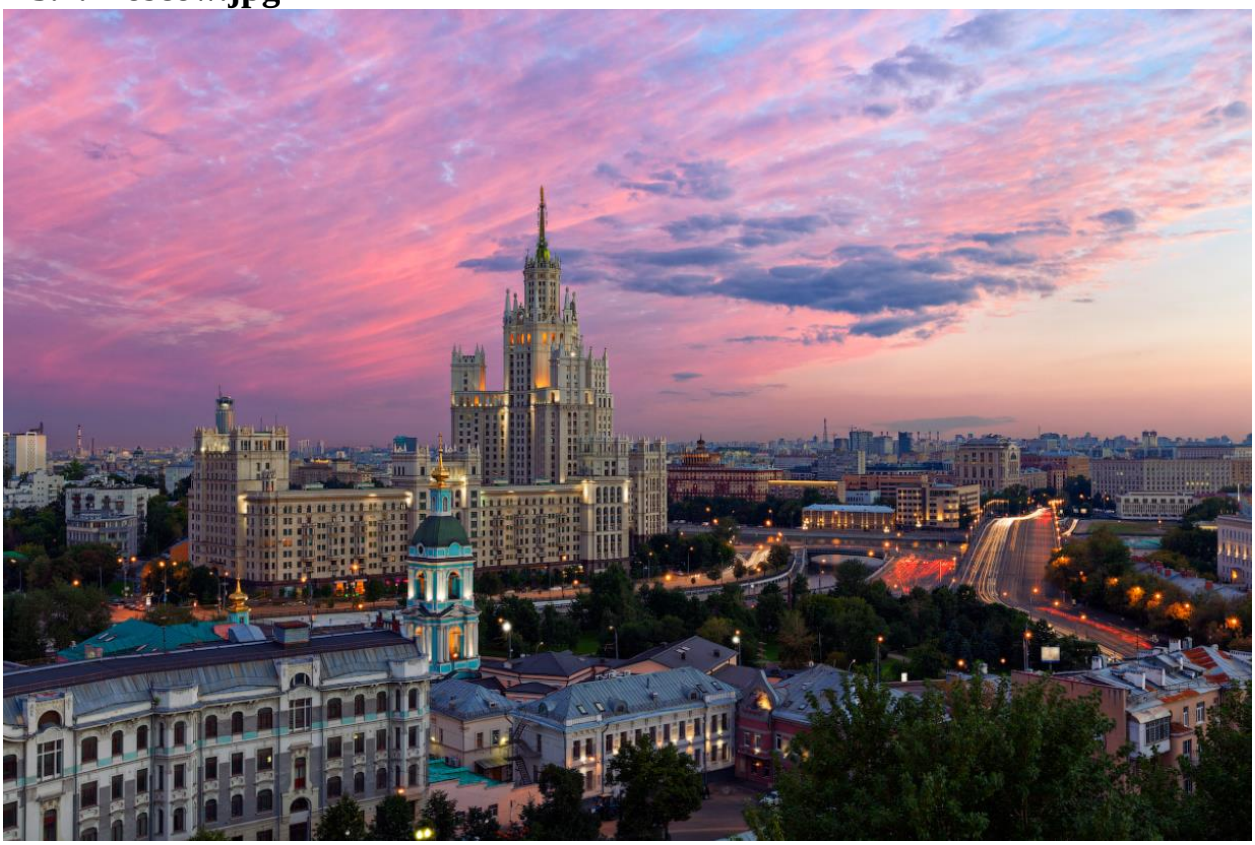
```
import json

file_locator = 'D:\Работа\МГТУ им.
Н.Э.Баумана\Программирование\Программы\Программы за 5
семестр\DZ\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'],
                            'reault': j['result']
                        }
                        data_1[id_user_data].append(n)
                    temp.append(data_1)
        data_recording(data_1)
    except:
        return 'Ошибка! Такого идентификатора не существует.'
```

3.6.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'
```

3.7.Moscow.jpg



Находится в папке «Function»:

3.8.__init__.py

```
print('Directory function')
```

3.9.Filed.py

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
 ]

def field(items, *args):
    try:
        # Преобразование в кортеж из строки
        argv = into_cortes_from_str(*args)
        # Необходимо реализовать генератор
        # len () возвращает длину (количество элементов) в объекте.
        assert len(argv) > 0, 'Ошибка! Нет аргументов! \nПримечание
аргументы не должны быть пустыми!'
        #range - диапазон, len - длина списка
        #items - это переменная, в которой на каждой итерации оказывается
элемент списка.
        n = [{} for i in range(len(items))]
        for i in range(len(items)):
            for j in items[i]:
                if j in argv:
                    #update - метод обновления словаря элементами из
другого объекта словаря.
```

```
                    n[i].update({j: items[i][j]})
        # возврат значения
        return n
    except:
        print('Ошибка! Нет списка в качестве переданного аргумента!')

# Преобразование в строку из кортежа
def into_cortes_from_str(str):
    cortes = []
    str_buf = ''
    for i in range(len(str)):
        if (str[i] == ' '):
            cortes.append(str_buf)
            str_buf = ''
        else:
            str_buf += str[i]
    # append - метод добавления элементов
    cortes.append(str_buf)
    # возврат значения
    return cortes
```

3.10. Unique.py

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.arr = []

        # Используя кортежи, получаем ключ и значения
        for key, meaning in kwargs.items():
            # Если ключ пустой и значение TRUE, то
            if key == 'ignore_case' and meaning == True:
                # Методы lower () возвращают строку в нижнем регистре из
заданной строки.
                # Он преобразует все заглавные символы в строчные.
                items = [i.lower() for i in items]

        for index in items:
            # Если текущее значение из списка item не совпадает/не
существует в созданном списке mas
            if index not in self.arr:
                # То присваиваем несуществующее значение в созданном
списке arr
                self.arr.append(index)
        pass

    # Для перехода к следующему элементу используется метод __next__.
    def __next__(self):
        try:
            x = self.arr[self.begin]
            self.begin += 1
            return x
        except:
            # Оператор raise позволяет принудительно породить исключение.
(Завершение работы итератора)
            raise StopIteration

    #__iter__(self) метод, который возвращает объект итератора;
    def __iter__(self):
        self.begin = 0
        return self
```

3.11. Unittest

Находится в папке «Module_test»:

3.11.1. __init.py__

```
print('Directory module_test')
```

3.11.2. Test_calculate.py

```
import unittest

from calculate.arithmetic_calculate import mathematical_calculator as smc

class test_calculate(unittest.TestCase):

    # Проверка на работу
    def test_1(self):
        self.assertEqual(smc('20.0').result, 20.0)
    def test_2(self):
        self.assertEqual(smc('12 + 12').result, 24.0)
    def test_3(self):
        self.assertEqual(smc('3 + 5 * 2').result, 13.0)
    def test_4(self):
        self.assertEqual(smc('5 + 2 + 9').result, 16.0)
    def test_5(self):
        self.assertEqual(smc('130 + 15 - 39').result, 106.0)
    def test_6(self):
        self.assertEqual(smc('5 * 2 / 5').result, 2.0)
    def test_7(self):
        self.assertEqual(smc('5 - 3 / 3').result, 4.0)
    def test_8(self):
        self.assertEqual(smc('5 * 0').result, 0.0)
    def test_9(self):
        self.assertEqual(smc('3 / 0').result, 'infinity')

if __name__ == '__main__':
    unittest.main()
```

3.11.3. Test_telebot.py

```
import unittest
import os.path

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

from calculate.work_calculate import generator_of_meaning,
get_info_id_user, data_recording

data_json_users_2 = {
    "745896123": [
        {"id": 61419,
         "meaning": 172,
         "result": 836.0},
        {"id": 3075,
         "meaning": "15 + 15",
         "result": "30.0"},
        {"id": 2878,
         "meaning": "15 + 15",
         "result": "30.0"},
        {"id": 6965,
         "meaning": "15 + 15",
         "result": "30.0"},
        {"id": 6409,
```

```
            "meaning": "10 / 0",
            "result": "infinity"},
    ],
    "965478145": [
        {"id": 6658,
         "meaning": "31 + 27",
         "result": "58.0"},
        {"id": 7427,
         "meaning": "142 + 440",
         "result": "582.0"},
        {"id": 9230,
         "meaning": "9 + 1 1",
         "result": "10.0"}
    ]
}

class test_telebot(unittest.TestCase):

    # Проверка создания файла
    def test_create_file(self):
        message_from_user_id = 745896123

        generator_of_meaning(str(message_from_user_id))

        self.assertEqual(
            os.path.exists(file_locator + '\data.json'),
            True
        )

    # Проверка на получение информации по id пользователя
    def test_get_info_id_user(self):
        data_recording(data_json_users_2)

        message_from_user_id = 745896123

        check_info = get_info_id_user(str(message_from_user_id))
        print(check_info)
        self.assertEqual(
            check_info, [{'id': 61419, 'meaning': 172, 'result': 836.0},
                         {'id': 3075, 'meaning': '15 + 15', 'result':
'30.0'},
                         {'id': 2878, 'meaning': '15 + 15', 'result':
'30.0'},
                         {'id': 6965, 'meaning': '15 + 15', 'result':
'30.0'},
                         {'id': 6409, 'meaning': '10 / 0', 'result':
'infinity'},
                         {'id': 6409, 'meaning': '10 / 0', 'result':
'infinity'}]
        )
```

3.11.4. Test_json.py

```
import unittest

from calculate.json_function import load_data, data_recording,
combined_data, load_data_for_id_user, delete_data_for_id_user

data_json = {
    "id_user": [
        {"id": 7581,
            "meaning": '10 + 50',
            "result": '60'}
    ]
```

```
}

data_json_big = {
    "id_user": [
        {"id": 6233,
            "meaning": '15 + 15',
            "result": '30'},
        {"id": 1665,
            "meaning": '15 + 15',
            "result": '30'},
        {"id": 6546,
            "meaning": '15 + 15',
            "result": '30'},
        {"id": 3959,
            "meaning": '15 + 15',
            "result": '30'},
        {"id": 4369,
            "meaning": '15 + 15',
            "result": '30'}
    ]
}

data_json1 = {
    "id_user": [
        {"id": 6855,
            "meaning": '103 + 102',
            "result": '205'}
    ]
}

data_json_with_id = {
    "103245678": [
        {"id": 7581,
            "meaning": '10 + 50',
            "result": '60'}
    ]
}

data_json_with_id_1 = {
    "103245678": [
        {"id": 9537,
            "meaning": '13 + 81',
            "result": '94'}
    ]
}

data_json_users_2 = {
    "745896123": [
            {"id": 61419,
             "meaning": 172,
             "result": 836.0},
            {"id": 3075,
             "meaning": "15 + 15",
             "result": "30.0"},
            {"id": 2878,
             "meaning": "15 + 15",
             "result": "30.0"},
            {"id": 6965,
             "meaning": "15 + 15",
             "result": "30.0"},
            {"id": 6409,
             "meaning": "10 / 0",
             "result": "infinity"},
```

```
    ],
      "965478145": [
            {"id": 6658,
              "meaning": "31 + 27",
              "result": "58.0"},
            {"id": 7427,
              "meaning": "142 + 440",
              "result": "582.0"},
            {"id": 9230,
              "meaning": "9 + 1 1",
              "result": "10.0"}
      ]
}

class test_json(unittest.TestCase):

    # Проверка на присутствия файла
    def test_write_and_read_file(self):
        # Создаем файл с данным
        data_recording(data_json)

        # Проверка наличия и сходимости
        self.assertEqual(
            load_data(),
            {'id_user': [{'id': 7581, 'result': '60', 'meaning': '10 +
50'}]}
        )

    # Проверка на добавлении json данных
    def test_append_json_in_json(self):
        # Создание файла с данными
        data_recording(data_json)

        # Изменение файла - добавление новых данных
        combined_data(data_json1)

        # Проверка наличия и сходимости
        self.assertEqual(
            load_data(),
            {'id_user': [
                {'id': 7581, 'result': '60', 'meaning': '10 + 50'},
                {'id': 6855, 'result': '205', 'meaning': '103 + 102'}
            ]})

    # Проверка на добавлении json данных с идентификатором пользователя
    def test_and_read_file_id(self):
        # Создание файла с данными
        data_recording(data_json_with_id)

        # Проверка наличия и сходимости
        self.assertEqual(
            load_data(),
            {'103245678': [{'id': 7581, 'result': '60', 'meaning': '10 +
50'}]}
        )

    # Проверка на добавлении json данных с идентификатором пользователя
    def test_append_json_in_json_id(self):
        # Создание файла с данными
        data_recording(data_json_with_id)

        # Изменение файла - добавление новых данных
        combined_data(data_json_with_id_1, str(103245678))
```

```
        # Проверка наличия и сходимости
        self.assertEqual(
            load_data(),
            {'103245678': [
                {'id': 7581, 'result': '60', 'meaning': '10 + 50'},
                {'id': 9537, 'result': '94', 'meaning': '13 + 81'}
            ]})
    #Проверка идентификатора поиска пользователя и получение информации
    def test_search_id_user_and_get_info(self):
        # Создание файл с данными
        data_recording(data_json_users_2)

        # Проверка наличия и сходимости
        self.assertEqual(
            load_data_for_id_user('965478145'),
            [{'id': 6658, 'result': '58.0', 'meaning': '31 + 27'},
             {'id': 7427, 'result': '582.0', 'meaning': '142 + 440'},
             {'id': 9230, 'result': '10.0', 'meaning': '9 + 1 1'},
             {'id': 9230, 'result': '10.0', 'meaning': '9 + 1 1'}])

    #Проверка удаления данных использования идентификатора
    def test_delete_data_of_id_user(self):
        # Создание файла с данными
        data_recording(data_json_users_2)

        # Удаление данные по id пользователя
        delete_data_for_id_user('745896123')

        # Проверка наличия и сходимости
        self.assertEqual(
            load_data_for_id_user('965478145'),
            'Ошибка! Такого идентификатора не существует.')
```

3.11.5. Test_filed.py

```
#Подключение библиотеки unittest для тестирования
import unittest

from function.filed import field, goods

#Создание класса тестирования filed
class test_filed(unittest.TestCase):
    # Проверка вывода с одним аргументом
    def test_check_output_one_argument(self):
        self.assertEqual(field(goods, 'title'),
            [
                {'title': 'Ковер'},
                {'title': 'Диван для отдыха'}
            ])

    # Проверка вывода с двумя аргументами
    def test_check_output_two_argument(self):
        self.assertEqual(field(goods, 'title color'),
            [
                {'color': 'green', 'title': 'Ковер'},
                {'color': 'black', 'title': 'Диван для отдыха'}
            ])

    # Проверка вывода с тремя аргументами
    def test_check_output_three_argument(self):
        self.assertEqual(field(goods, 'title color price'),
            [
                {'color': 'green', 'price': 2000, 'title': 'Ковер'},
```

```
                {'color': 'black', 'price': 5300, 'title': 'Диван для
отдыха'}
            ])
```

3.11.6. Test_unique.py

```
#Подключение библиотеки unittest для тестирования
import unittest

from function.unique import Unique

#Создание класса тестирования unique
class test_unique(unittest.TestCase):
    #Проверка на значения
    def test_check_meaning(self):
        meaning = [1, 1, 2, 2, 2, 3, 3, 3, 4, 4]
        #Получение уникальных элементов числового типа
        mas_unique = Unique(meaning).arr
        #Проверка
        self.assertEqual(mas_unique, [1, 2, 3, 4])

    #Проверка на буквы
    def test_check_symbol(self):
        symbol = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        #Получение уникальных элементов символьного типа
        mas_unique = Unique(symbol).arr
        #Проверка
        self.assertEqual(mas_unique, ['a', 'A', 'b', 'B'])

    #Проверка на буквы без чувствительного регистра
    def test_check_symbol_sensitive_register(self):
        symbol = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        #Получение уникальных элементов символьного типа
        mas_unique = Unique(symbol, ignore_case = True).arr
        #Проверка
        self.assertEqual(mas_unique, ['a', 'b'])

    #Проверка на буквы со значениями (смещанный тип)
    def test_check_symbol_meaning(self):
        sym_men = ['a', 'A', 'b', 'B', '1', '1', '2', '2']
        #Получение уникальных элементов смещанного типа
        mas_unique = Unique(sym_men).arr
        #Проверка
        self.assertEqual(mas_unique, ['a', 'A', 'b', 'B', '1', '2'])

if __name__ == '__main__':
    unittest.main()
```

3.12. Behave

Находится в папке «feature»:

3.12.1. __init__.py

```
print('Directory feature')
```

3.12.2. Filed.feature

```
Feature: Checking the output of the argument from the list of goods


  # Проверка вывода данных с 1 аргументом
  Scenario Outline: Checking data output with one argument
    Given There is a list of goods
    When Enter <arguments> to get meanings
```

```
    Then Output the meanings of <result>

    Examples:
      | arguments | result                                                    |
      | title     | [{'title': 'Ковер'},{'title': 'Диван для отдыха'}]        |
      | color     | [{'color': 'green'},{'color': 'black'}]                   |
      | price     | [{'price': 2000},{'price': 5300}]                         |


  # Проверка вывода данных с 2 аргументами
  Scenario Outline: Checking the output with two arguments
    Given There is a list of goods
    When Enter <arguments> to get meanings
    Then Output the meanings of <result>

    Examples:
      | arguments    | result
|
      | title color | [{'title': 'Ковер', 'color': 'green'},{'title':
'Диван для отдыха', 'color': 'black'}] |
      | color price | [{'color': 'green', 'price': 2000},{'color':
'black', 'price': 5300}]                 |


  # Проверка вывода данных с 3 аргументами
  Scenario Outline: Checking the output with three arguments
    Given There is a list of goods
    When Enter <arguments> to get meanings
    Then Output the meanings of <result>

    Examples:
      | arguments          | result
|
      | title color price | [{'color': 'green', 'price': 2000, 'title':
'Ковер'}, {'color': 'black', 'price': 5300, 'title': 'Диван для отдыха'}]
|
```

3.12.3. Unique.feature

```
Feature: Calculating and obtaining unique meanings

  # Уникальные элементы числового типа
  Scenario Outline: Get unique meaning from a list of numbers
    Given There is a class of unique meanings
    And Get the list: <LIST>
    When Search for unique elements using: <CASE>
    Then Output unique elements: <UNIQUE>

    Examples:
      | LIST                          | UNIQUE       | CASE |
      | [1, 1, 2, 2, 2, 3, 3, 3, 4, 4] | [1, 2, 3, 4] | 0    |
      | [2, 3, 1, 1, 5, 3, 2, 5, 2, 1] | [2, 3, 1, 5] | 0    |
      | [1, 1, 1, 1, 3, 1, 2, 1, 2, 1] | [1, 3, 2]    | 0    |

  # Уникальные элементы символьного типа
  Scenario Outline: Get unique meaning from a list of symbols
    Given There is a class of unique meanings
    And Get the list: <LIST>
    When Search for unique elements using: <CASE>
    Then Output unique elements: <UNIQUE>

    Examples:
      | LIST                                        | UNIQUE
| CASE |
```

```
      | ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'] | ['a', 'A', 'b', 'B']
| 0    |
      | ['c', 'C', 'b', 'A', 'C', 'a', 'c', 'B'] | ['c', 'C', 'b', 'A',
'a', 'B'] | 0    |
      | ['n', 'D', 'm', 'N', 'd', 'n', 'M', 'N'] | ['n', 'D', 'm', 'N',
'd', 'M'] | 0    |


  # Уникальные элементы символьного типа без чувствительного регистра
     # Если <CASE> равен 1, то это верно
  Scenario Outline: Get unique meaning from the ignore_case symbol list
    Given There is a class of unique meanings
    And Get the list: <LIST>
    When Search for unique elements using: <CASE>
    Then Output unique elements: <UNIQUE>

    Examples:
      | LIST                                     | UNIQUE          | CASE
|
      | ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'] | ['a', 'b']      | 1
|
      | ['c', 'C', 'b', 'A', 'C', 'a', 'c', 'B'] | ['c', 'b', 'a'] | 1
|


  # Уникальные элементы смешанного типа
     # Если <CASE> равен 1, то это верно
  Scenario Outline: Get unique meanings from a mixed type list
    Given There is a class of unique meanings
    And Get the list: <LIST>
    When Search for unique elements using: <CASE>
    Then Output unique elements: <UNIQUE>

    Examples:
      | LIST                                     | UNIQUE
| CASE |
      | ['a', 'A', 'b', 'B', '1', '1', '2', '2'] | ['a', 'A', 'b', 'B',
'1', '2'] | 0    |
      | ['a', 'A', 'b', 'B', '1', '1', '2', '2'] | ['a', 'b', '1', '2']
| 1    |
```

3.13. Steps

Находится в папке «feature» - > «steps»:

3.13.1. __init__.py

```
print('Directory steps')
```

3.13.2. Filed.py (для steps)

```
from behave import Given, When, Then
from function.filed import field, goods
import ast

@Given('There is a list of goods')
def for_given(text):
    text.data_dictonary = goods
    test = text.data_dictonary
    print(test)

@When("Enter {arguments} to get meanings")
def for_when(text, arguments):
    text.results = field(text.data_dictonary, arguments)
```

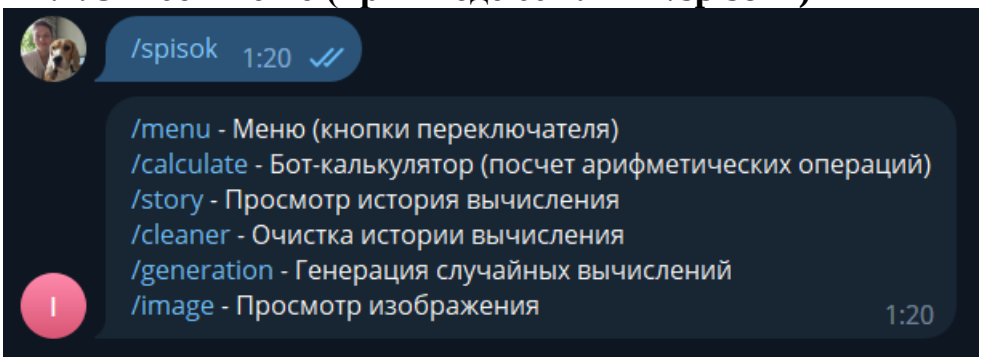
```
@Then("Output the meanings of {result}")
def for_then(text, result):
    assert text.results == ast.literal_eval(result)
```

3.13.3. Unique.py (для steps)

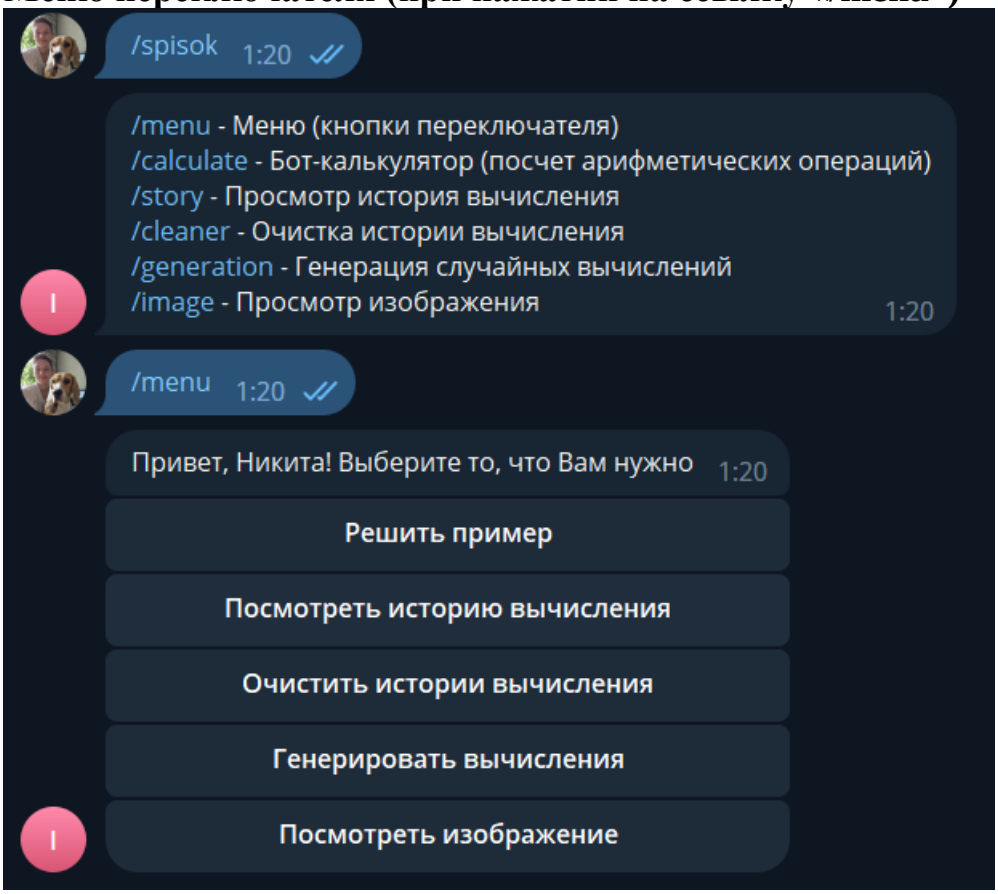
```
from behave import Given, When, Then
from function.unique import Unique
import ast

@Given('There is a class of unique meanings')
def for_given(text):
    pass

@Given("Get the list: {LIST}")
def for_given_and(text, LIST):
    text.LIST = list(ast.literal_eval(LIST))
    print(f'Список: {LIST}')

@When("Search for unique elements using: {CASE}")
def for_when(text, CASE):
    check = bool(int(CASE))
    if (check == True):
        unique_list = Unique(text.LIST, ignore_case=check)
    else:
        unique_list = Unique(text.LIST)
    text.results = unique_list

@Then("Output unique elements: {UNIQUE}")
def for_then(text, UNIQUE):
    assert text.results.arr == ast.literal_eval(UNIQUE)
    print(f'Уникальные элементы: {text.results.arr}')
```

4. Результат работы программы

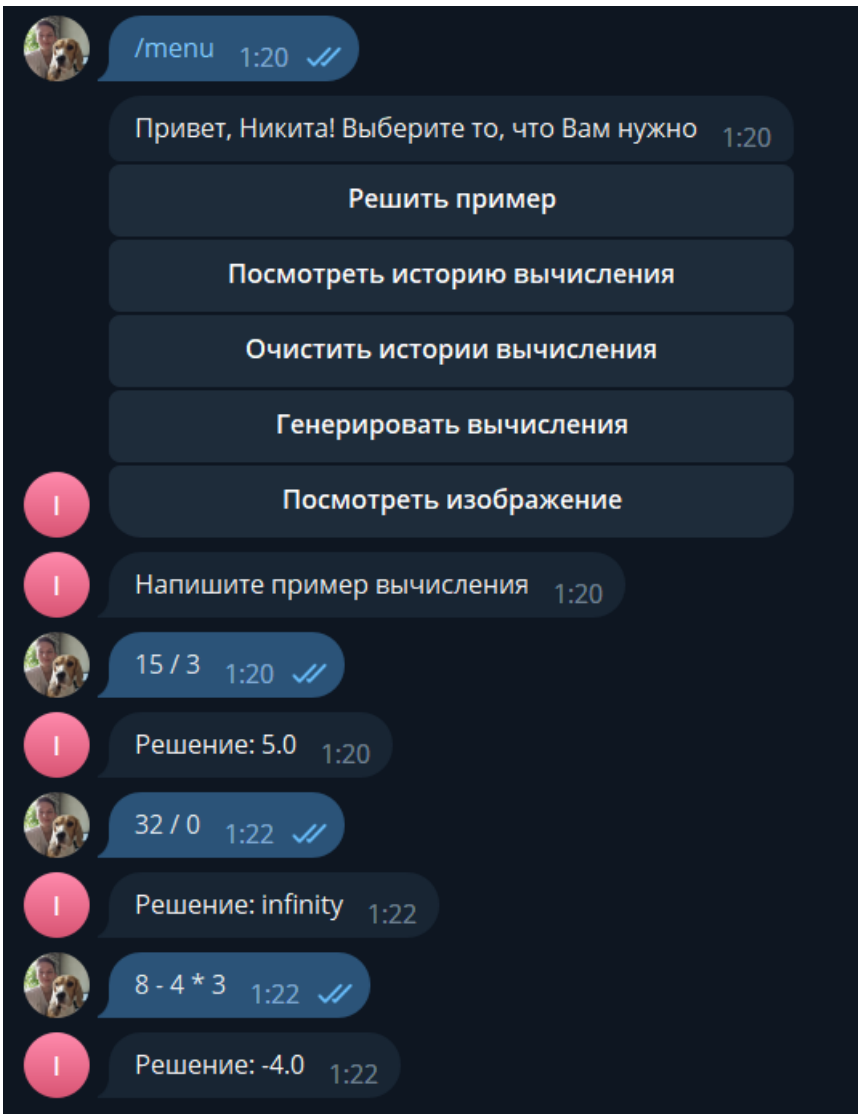
4.1.Список меню (при вводе ссылки «/spisok»)



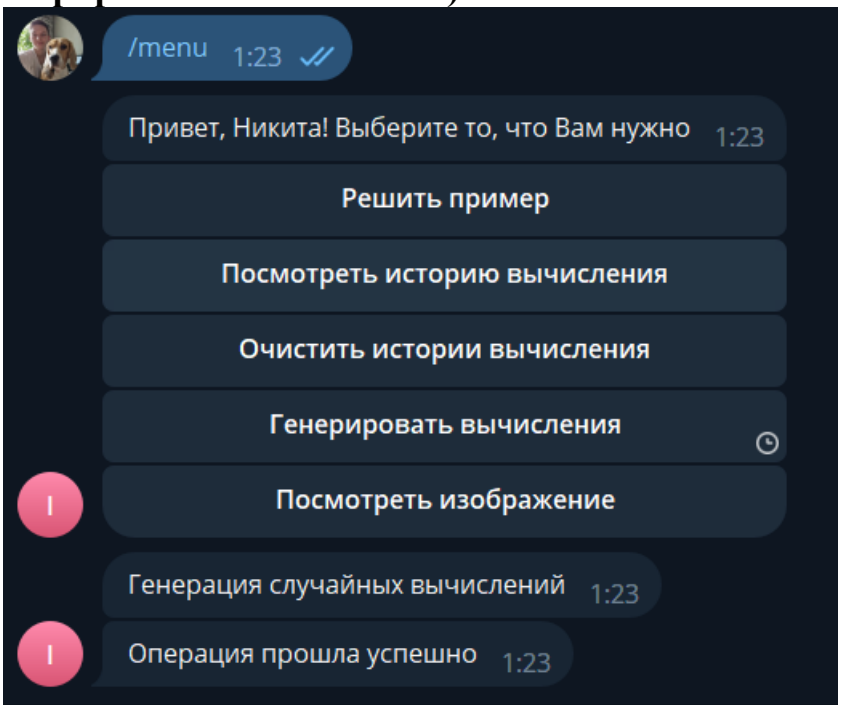
Меню переключателя (при нажатии на ссылку «/menu»)



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



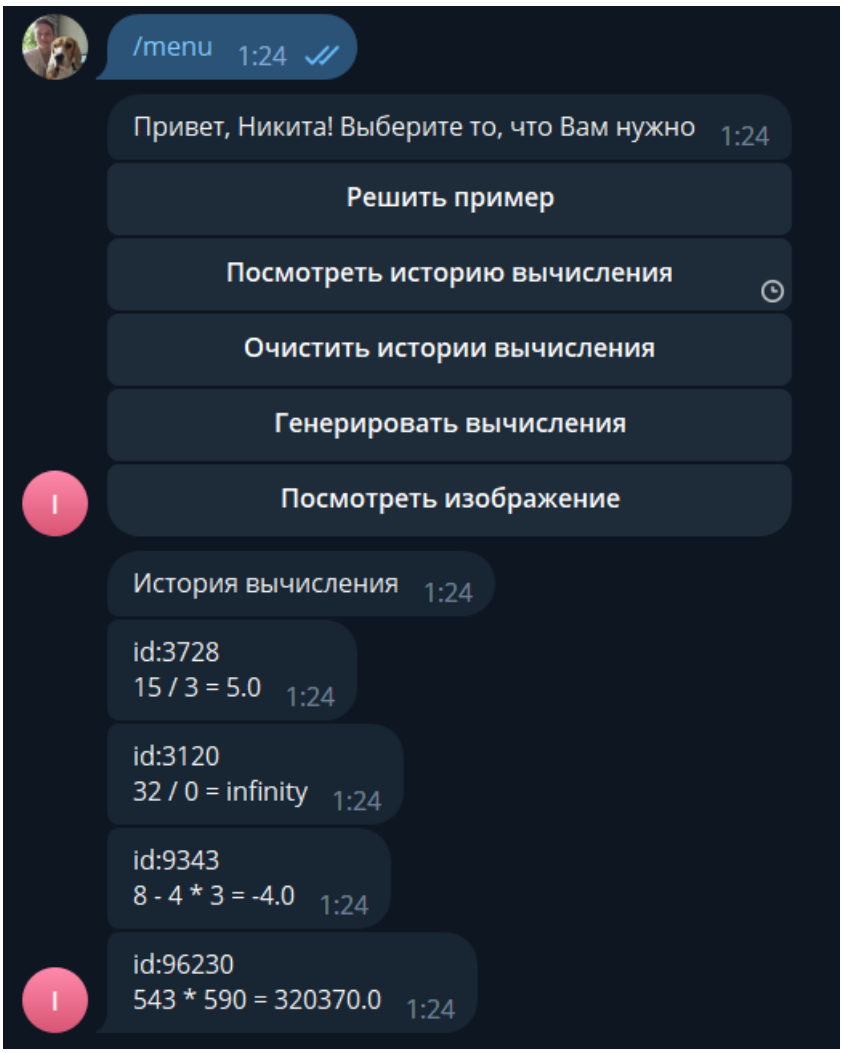
Генерация случайных вычислений (при нажатии на кнопку «Генерировать вычисления»)



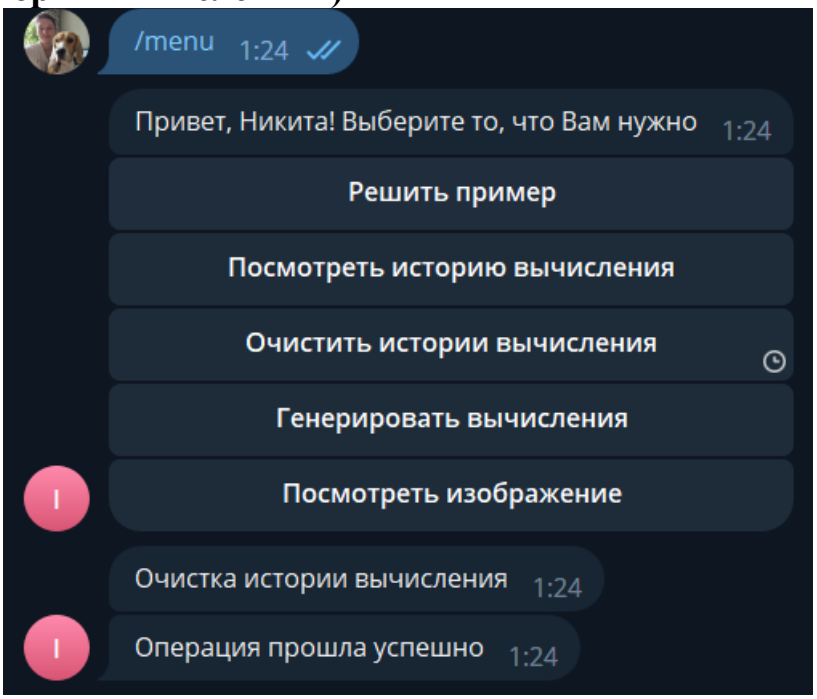
Данные хранятся в data.json

Вместо “***id_user***” должны быть цифры индекса пользователя из телеграмма.

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

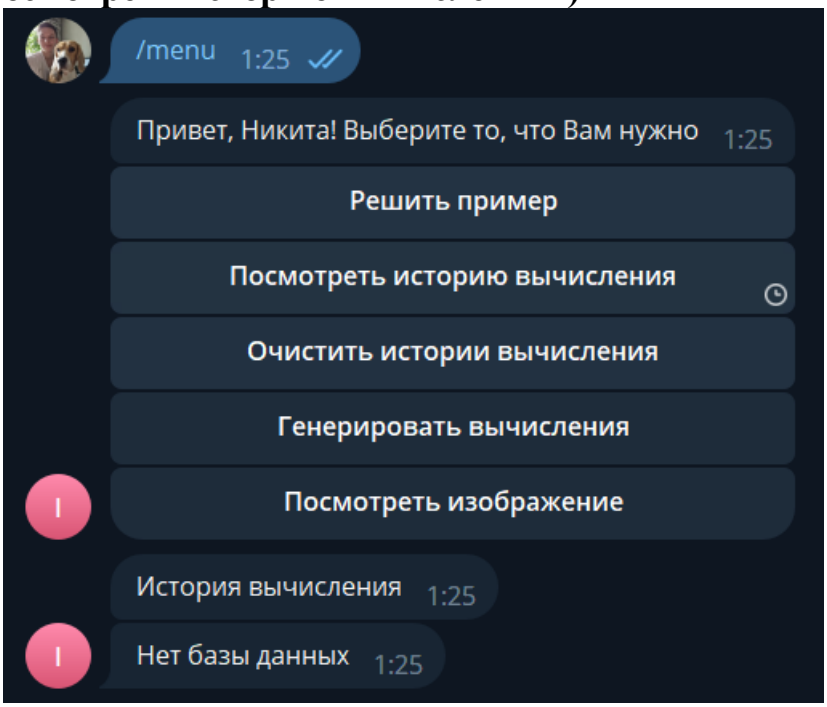


Очистка истории вычисления (при нажатии на кнопку «Очистить истории вычисления»)

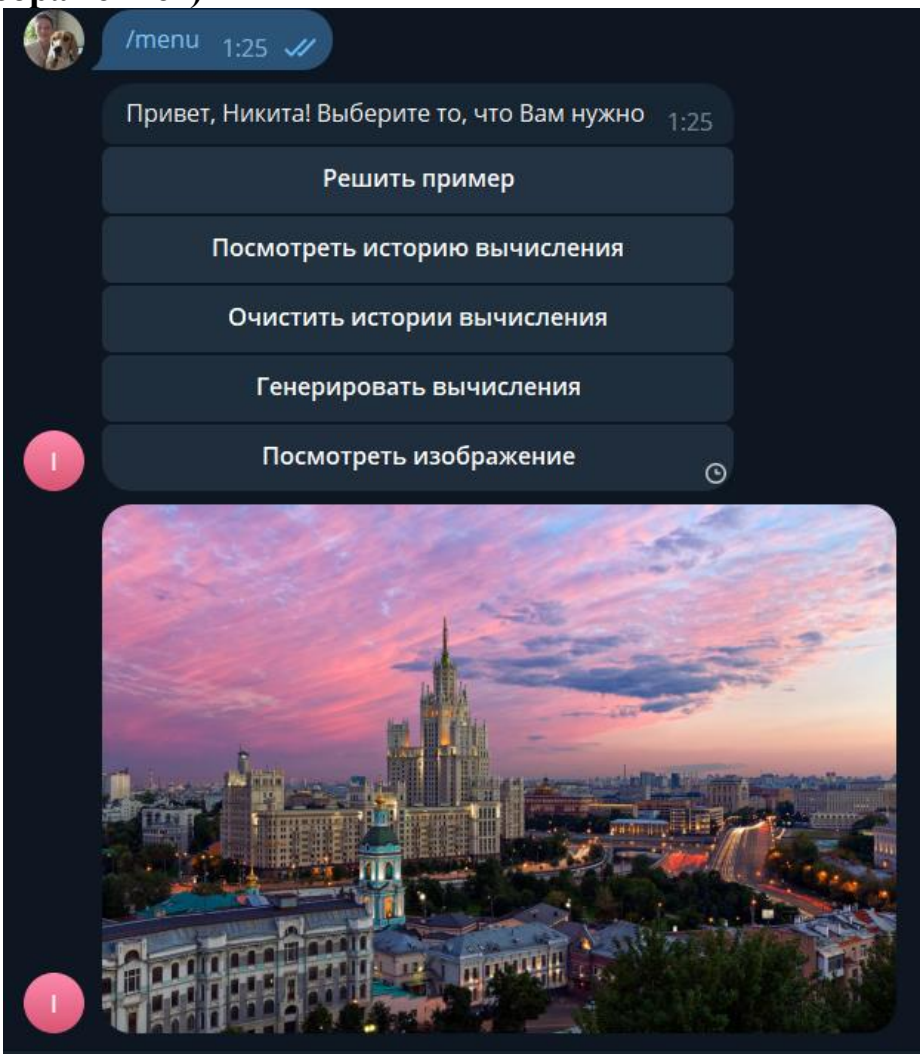


Проверка, очистила ли история вычисления (при нажатии на кнопку

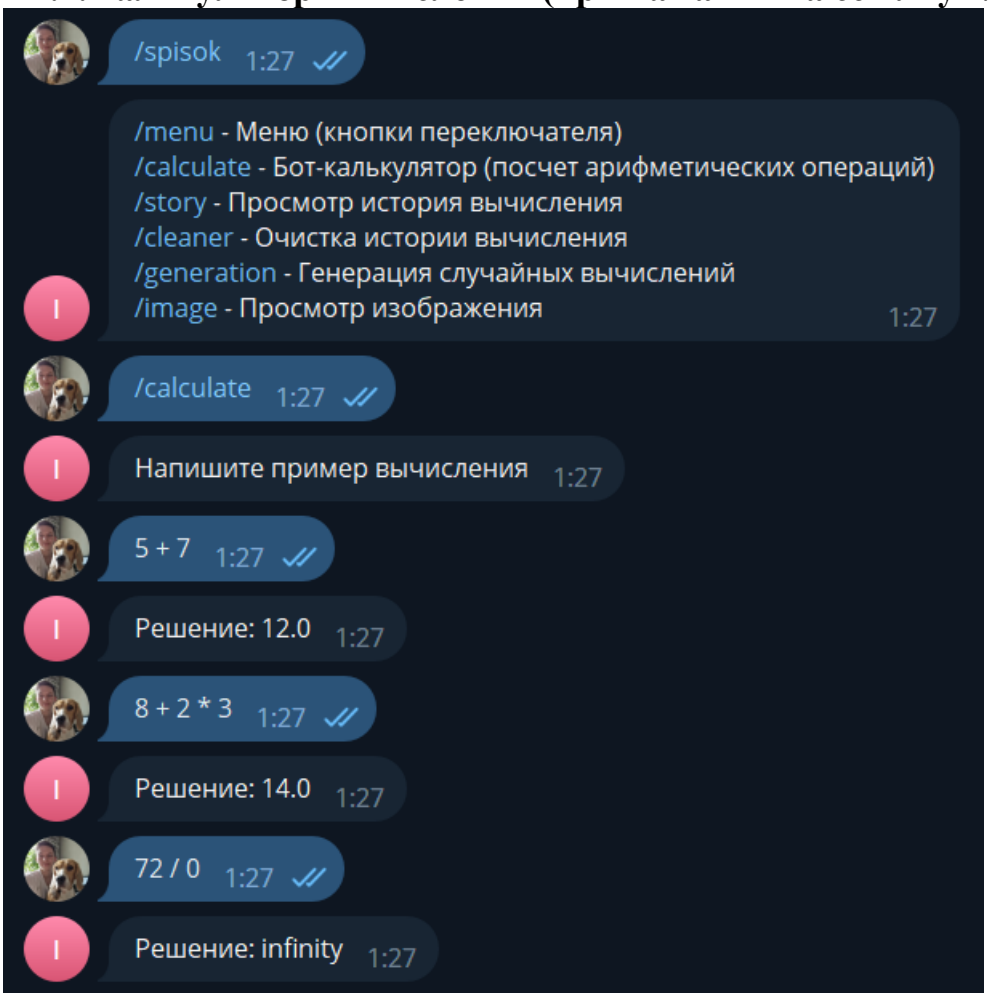
«Посмотреть историю вычисления»)



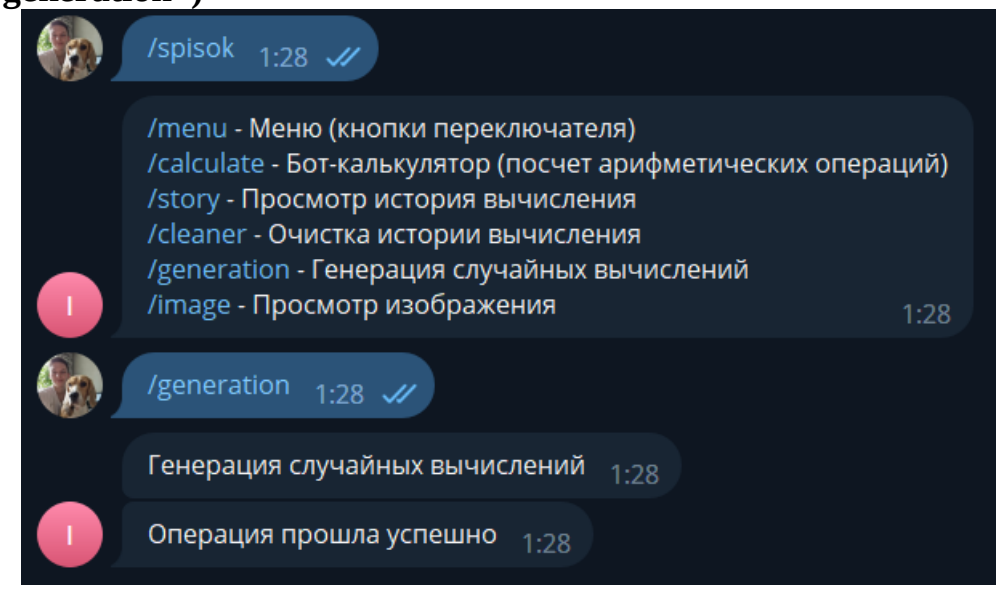
Просмотр изображения (при нажатии на кнопку «Посмотреть изображение»)



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

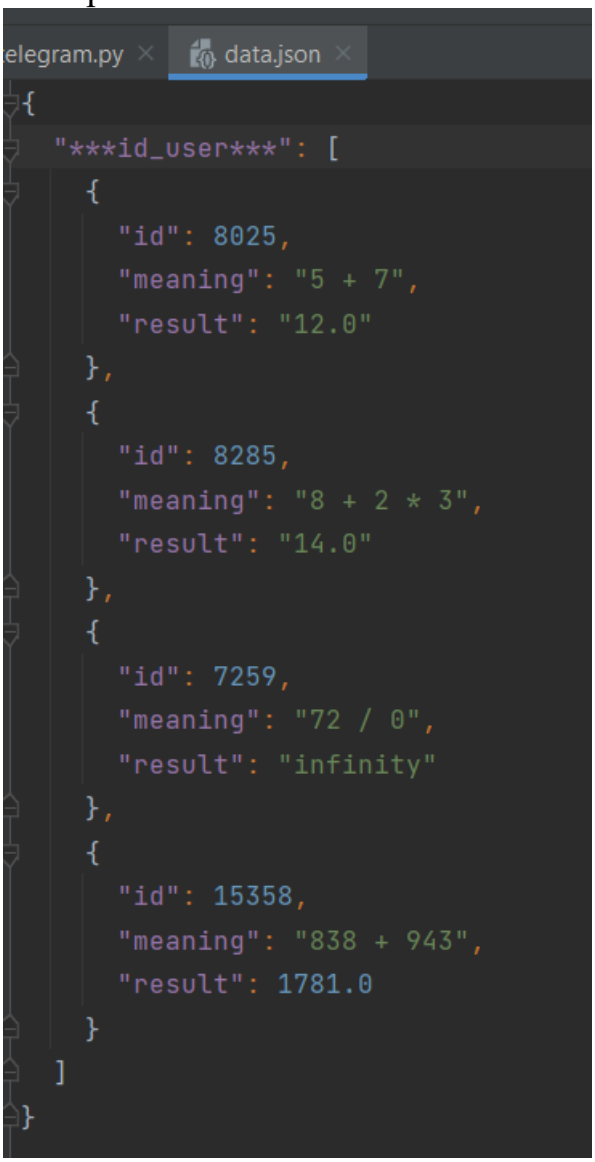


Генерация случайных вычислений (при нажатии на ссылку «/generation»)

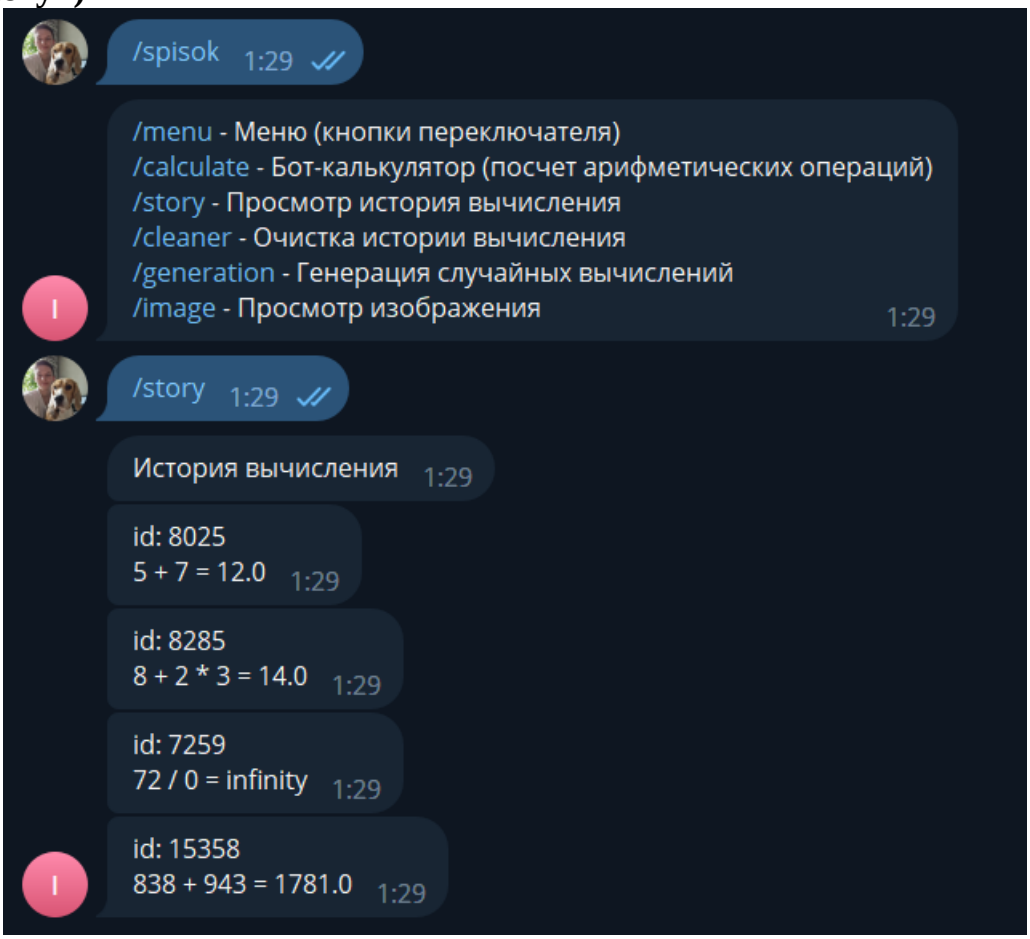


Данные хранятся в data.json

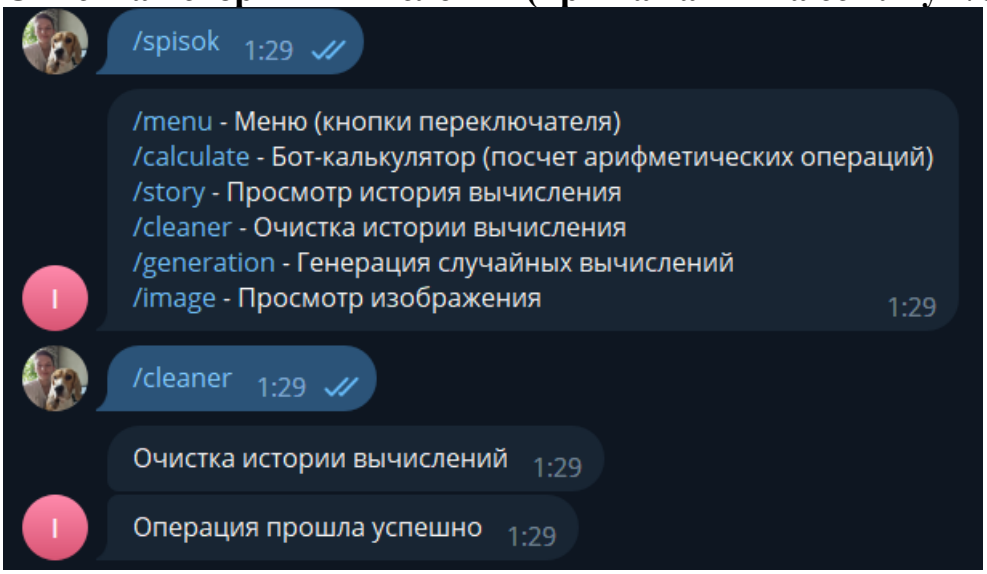
Вместо “***id_user***” должны быть цифры индекса пользователя из телеграмма.



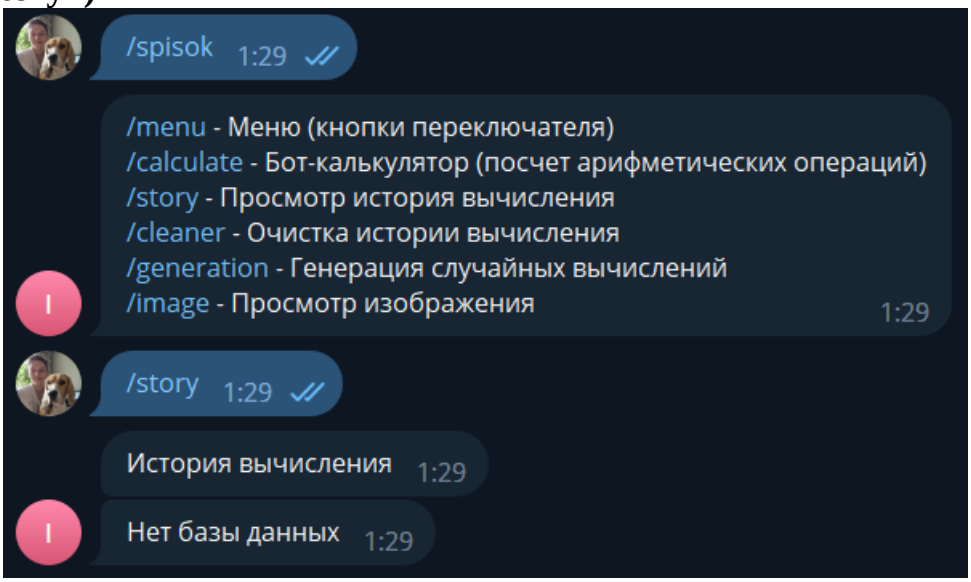
Чтение и просмотр данные из data.json (при нажатии на ссылку «story»)



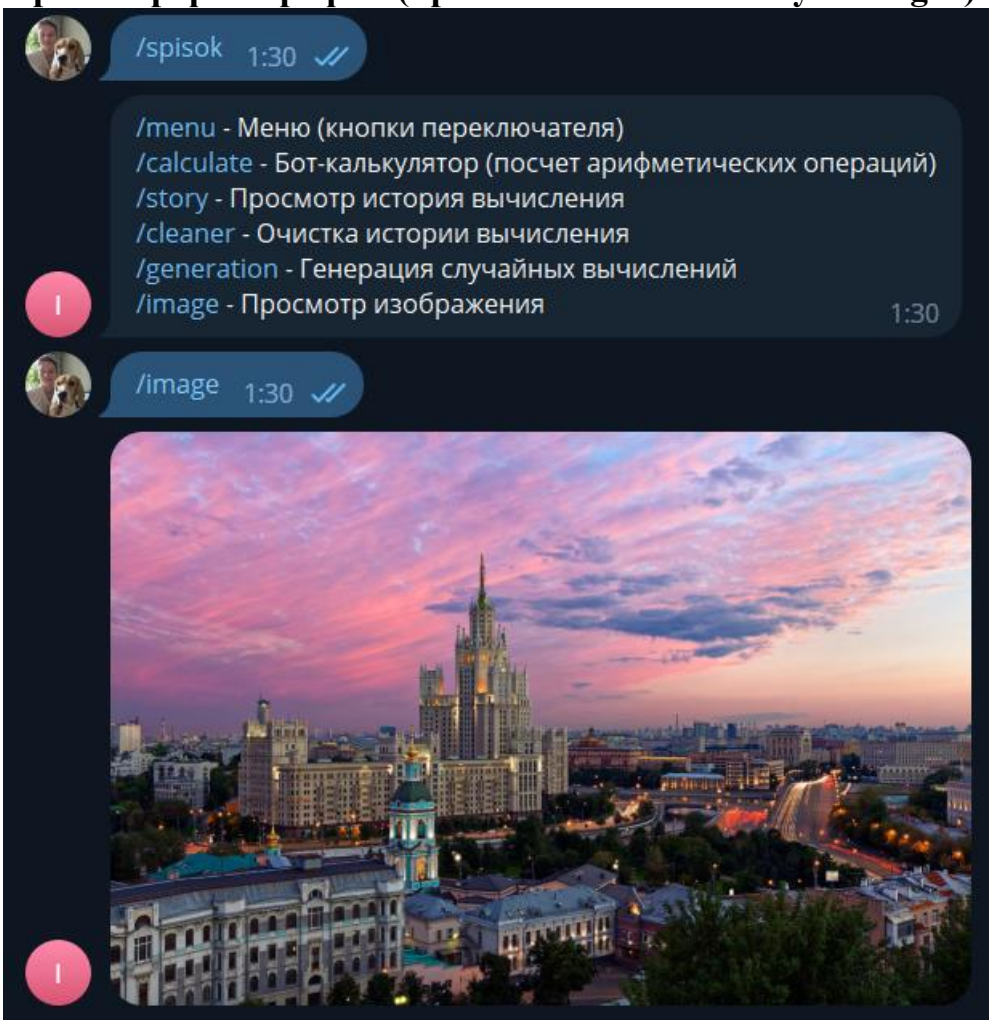
Очистка истории вычисления (при нажатии на ссылку «/cleaner»)



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



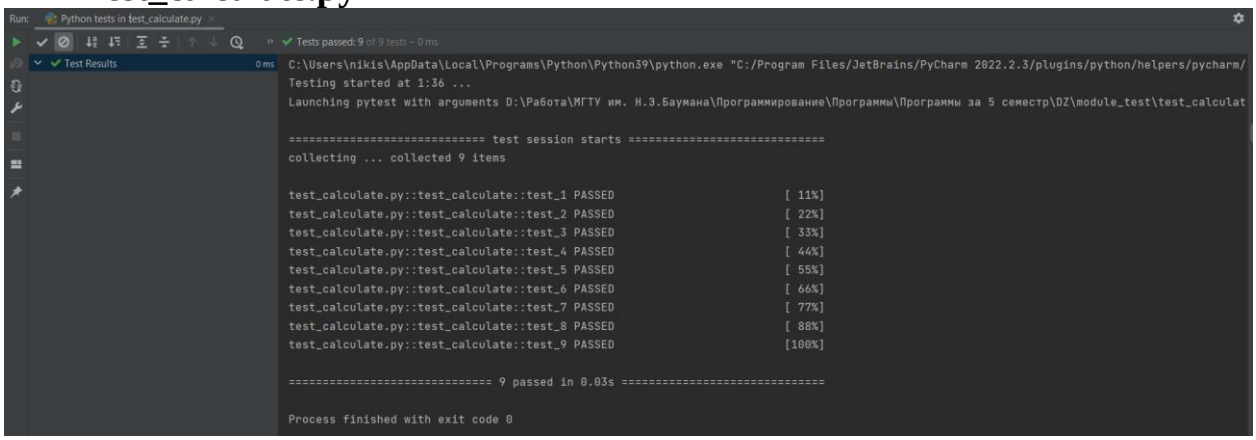
Просмотр фотографии (при нажатии на ссылку «/image»)



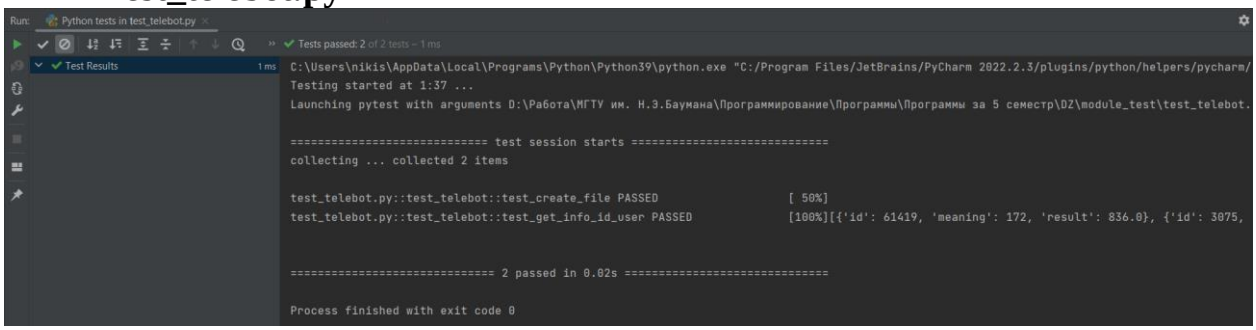
5. Модульное тестирование

5.1.Unittest

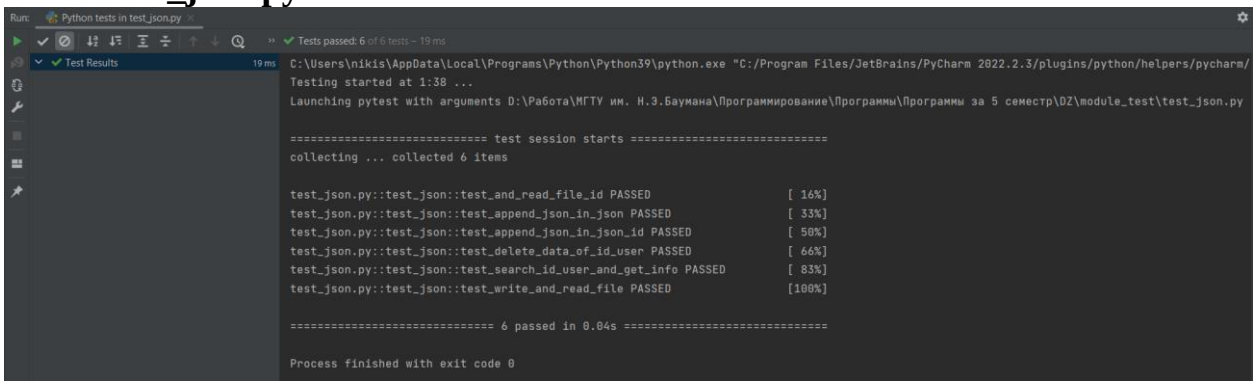
Test_calculate.py



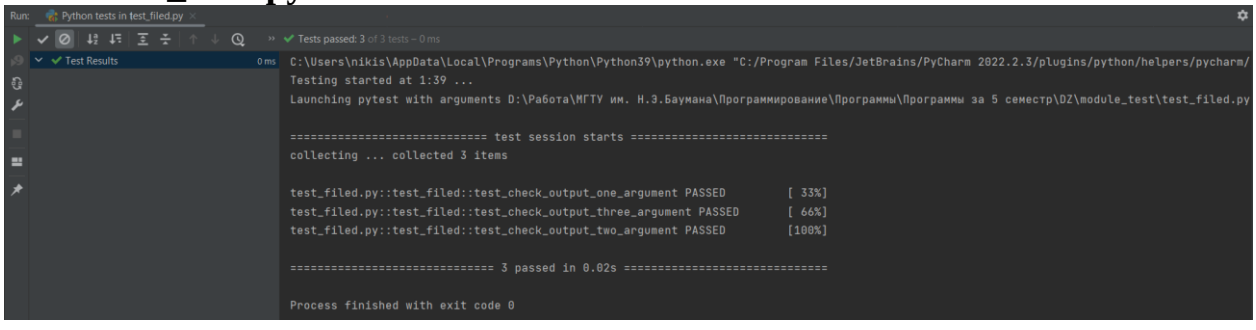
Test_telebot.py



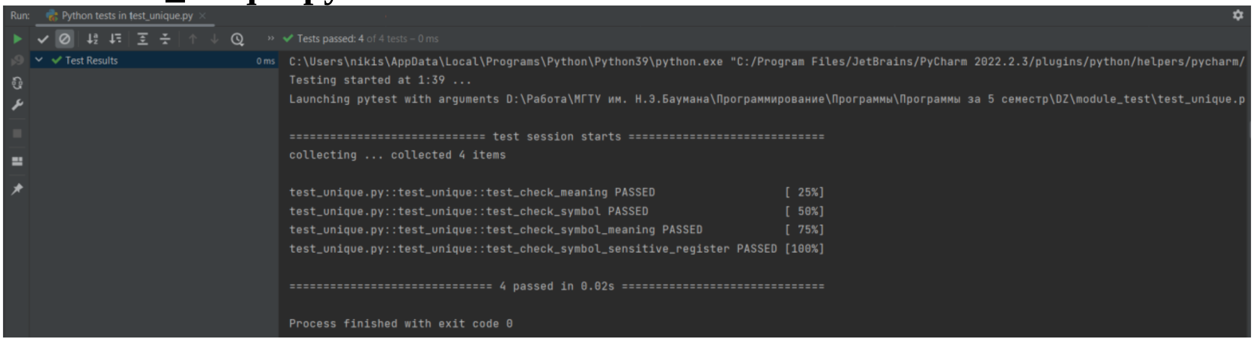
Test_json.py



Test_filed.py



Test_unique.py



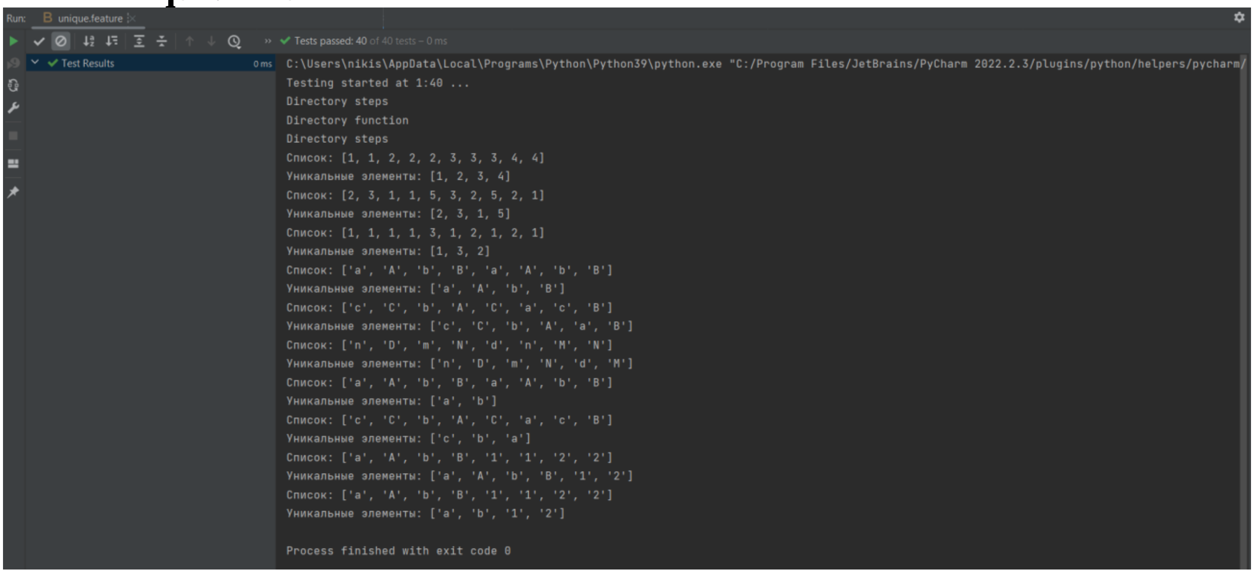
```
C:\Users\nikis\AppData\Local\Programs\Python\Python39\python.exe "C:/Program Files/JetBrains/PyCharm 2022.2.3/plugins/python/helpers/pycharm/
Testing started at 1:39 ...
Launching pytest with arguments D:\Работа\МГТУ им. Н.З.Баумана\Программирование\Программы\Программы за 5 семестр\DZ\module_test\test_unique.p

============================ test session starts ============================
collecting ... collected 4 items

test_unique.py::test_unique::test_check_meaning PASSED                   [ 25%]
test_unique.py::test_unique::test_check_symbol PASSED                    [ 50%]
test_unique.py::test_unique::test_check_symbol_meaning PASSED            [ 75%]
test_unique.py::test_unique::test_check_symbol_sensitive_register PASSED [100%]

============================= 4 passed in 0.02s =============================

Process finished with exit code 0
```

5.2.Behave

Filed.feature

```
C:\Users\nikis\AppData\Local\Programs\Python\Python39\python.exe "C:/Program Files/JetBrains/PyCharm 2022.2.3/plugins/python/helpers/pycharm/
Testing started at 1:40 ...
Directory steps
Directory function
Directory steps
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]
[{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]

Process finished with exit code 0
```

Unique.feature



```
C:\Users\nikis\AppData\Local\Programs\Python\Python39\python.exe "C:/Program Files/JetBrains/PyCharm 2022.2.3/plugins/python/helpers/pycharm/
Testing started at 1:40 ...
Directory steps
Directory function
Directory steps
Список: [1, 1, 2, 2, 2, 3, 3, 3, 4, 4]
Уникальные элементы: [1, 2, 3, 4]
Список: [2, 3, 1, 1, 5, 3, 2, 5, 2, 1]
Уникальные элементы: [2, 3, 1, 5]
Список: [1, 1, 1, 1, 3, 1, 2, 1, 2, 1]
Уникальные элементы: [1, 3, 2]
Список: ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
Уникальные элементы: ['a', 'A', 'b', 'B']
Список: ['c', 'C', 'b', 'A', 'C', 'a', 'c', 'B']
Уникальные элементы: ['c', 'C', 'b', 'A', 'a', 'B']
Список: ['n', 'D', 'm', 'N', 'd', 'n', 'M', 'N']
Уникальные элементы: ['n', 'D', 'm', 'N', 'd', 'M']
Список: ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
Уникальные элементы: ['a', 'b']
Список: ['c', 'C', 'b', 'A', 'C', 'a', 'c', 'B']
Уникальные элементы: ['c', 'b', 'a']
Список: ['a', 'A', 'b', 'B', '1', '1', '2', '2']
Уникальные элементы: ['a', 'A', 'b', 'B', '1', '2']
Список: ['a', 'A', 'b', 'B', '1', '1', '2', '2']
Уникальные элементы: ['a', 'b', '1', '2']

Process finished with exit code 0
```