|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 3](#_Toc121789946)

[2. Описание задания 3](#_Toc121789947)

[3. Текст программы 4](#_Toc121789948)

[4. Результат работы программы 22](#_Toc121789949)

[5. Модульное тестирование 31](#_Toc121789950)

## Цель домашнего задания

Изучение возможностей создания ботов в Telegram и их тестирования.

## 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).

## Текст программы

Находится в папке «Calculate»:

* 1. **Link.py**

Здесь должен быть токен для доступа к HTTP API

token = ''

* 1. **\_\_init\_\_.py**

print('Directory calculate')

* 1. **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

* 1. **Bot\_telegram.py**

import link  
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(link.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)

* 1. **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.'

* 1. **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'

* 1. **Moscow.jpg**

****

Находится в папке «Function»:

* 1. **\_\_init\_\_.py**

print('Directory function')

* 1. **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:  
 #updatе - 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

* 1. **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. (Заверщение работы итератора)  
 raise StopIteration  
  
 #\_\_iter\_\_(self) метод, который возвращает объект итератора;  
 def \_\_iter\_\_(self):  
 self.begin = 0  
 return self

* 1. **Unittest**

Находится в папке «Module\_test»:

* + 1. **\_\_init.py\_\_**

print('Directory module\_test')

* + 1. **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()

* + 1. **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'}]  
 )

* + 1. **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.')

* + 1. **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'}  
 ])

* + 1. **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()

* 1. **Behave**

Находится в папке «feature»:

* + 1. **\_\_init\_\_.py**

print('Directory feature')

* + 1. **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'}]** |

* + 1. **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** |

* 1. **Steps**

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

* + 1. **\_\_init\_\_.py**

print('Directory steps')

* + 1. **Filed.py (для steps)**

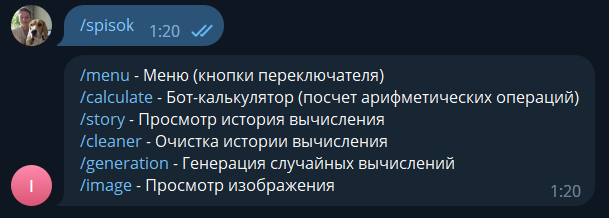
from behave import Given, When, Then  
from function.filed import field, goods  
import ast  
  
@Given('There is a list of goods')  
def for\_given(text):  
 text.data\_dictonary = goods  
 test = text.data\_dictonary  
 print(test)  
  
@When("Enter {arguments} to get meanings")  
def for\_when(text, 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)

* + 1. **Unique.py (для steps)**

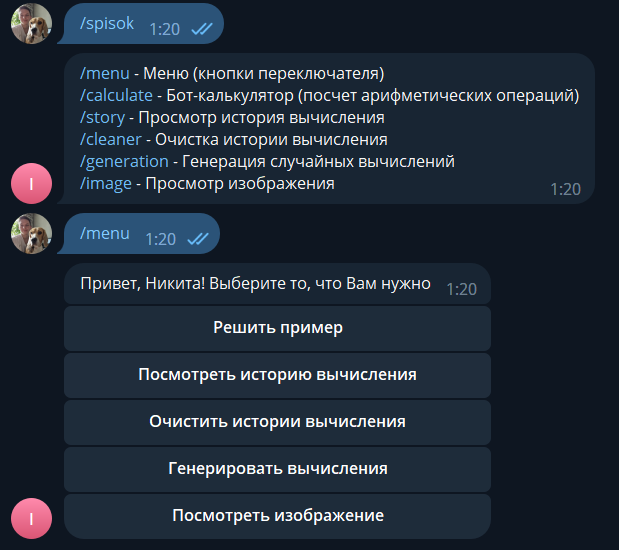
from behave import Given, When, Then  
from function.unique import Unique  
import ast  
  
@Given('There is a class of unique meanings')  
def for\_given(text):  
 pass  
  
@Given("Get the list: {LIST}")  
def for\_given\_and(text, LIST):  
 text.LIST = list(ast.literal\_eval(LIST))  
 print(f'Список: {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}')

## Результат работы программы

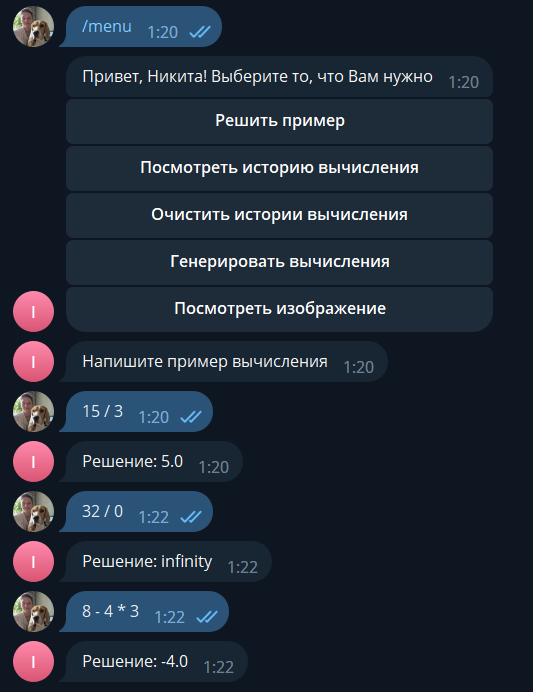
* 1. **Список меню (при вводе ссылки «/spisok»)**



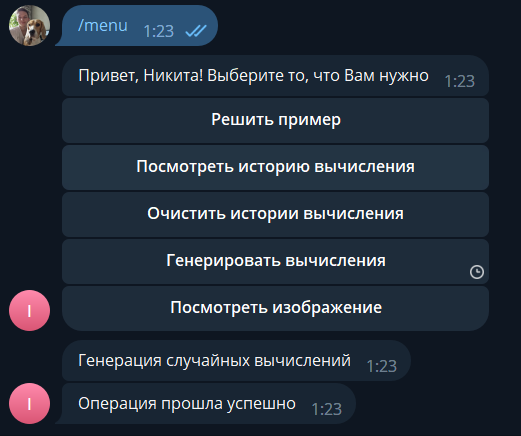
**Меню переключателя (при нажатии на ссылку «/menu»)**



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

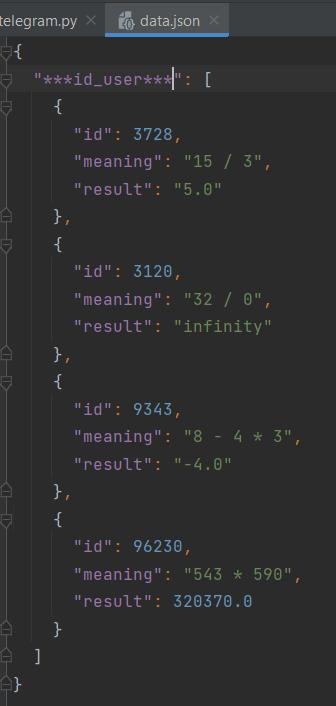


**Генерация случайных вычислений (при нажатии на кнопку «Генерировать вычисления»)**

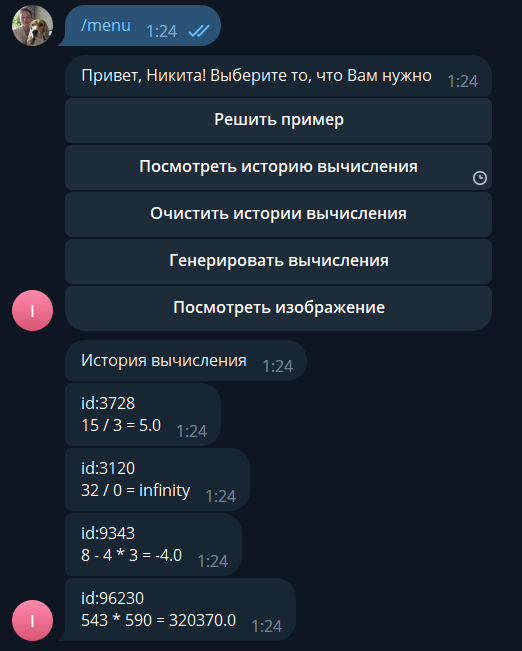


**Данные хранятся в data.json**

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



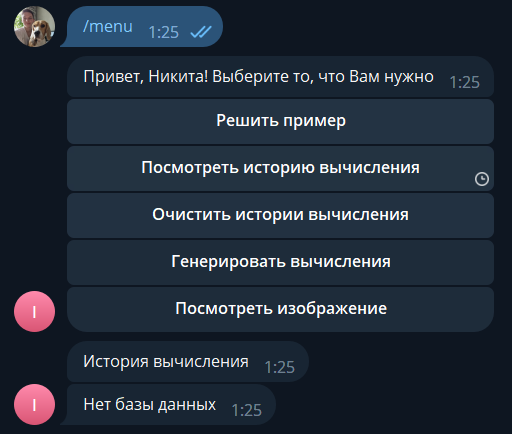
**Чтение и просмотр данные из data.json (при нажатии на кнопку «Посмотреть историю вычисления»)**



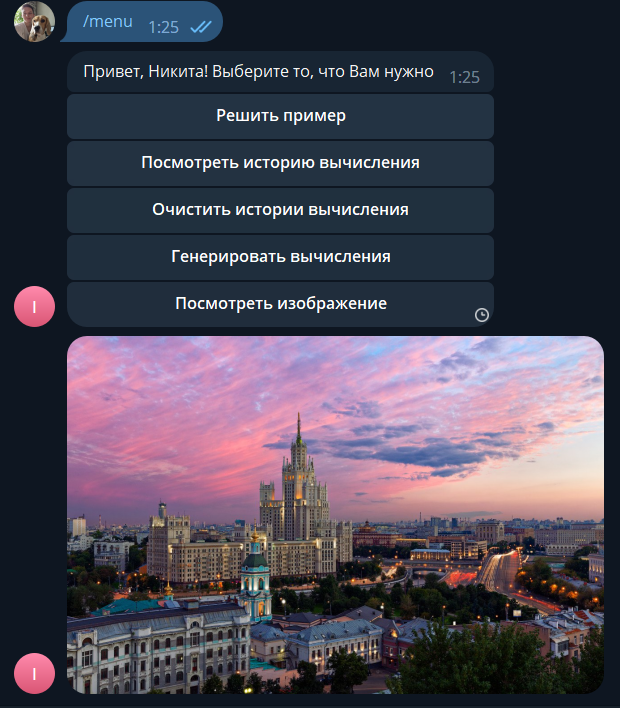
**Очистка истории вычисления (при нажатии на кнопку «Очистить истории вычисления»)**

****

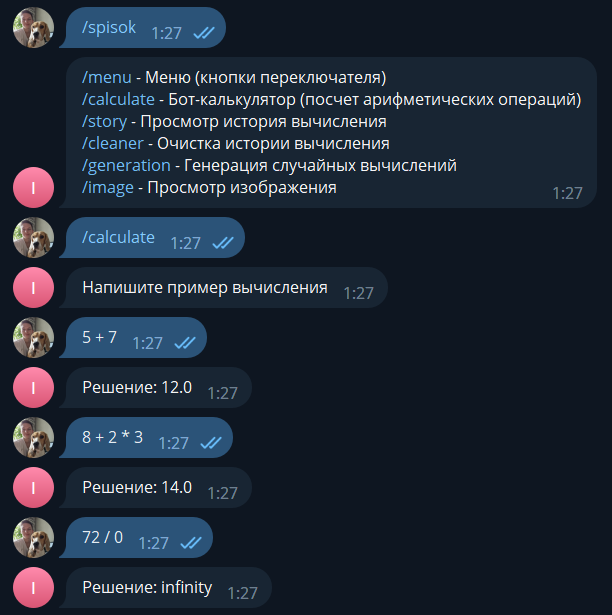
**Проверка, очистила ли история вычисления (при нажатии на кнопку «Посмотреть историю вычисления»)**



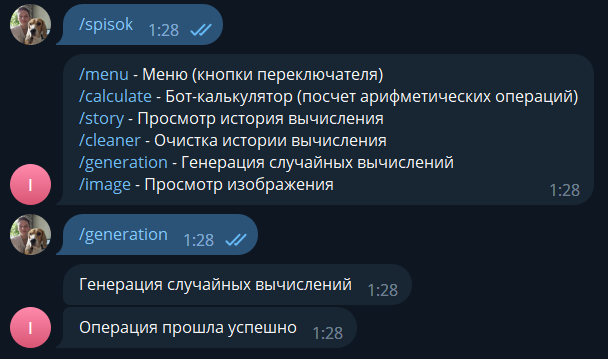
**Просмотр изображения (при нажатии на кнопку «Посмотреть изображение»)**

****

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

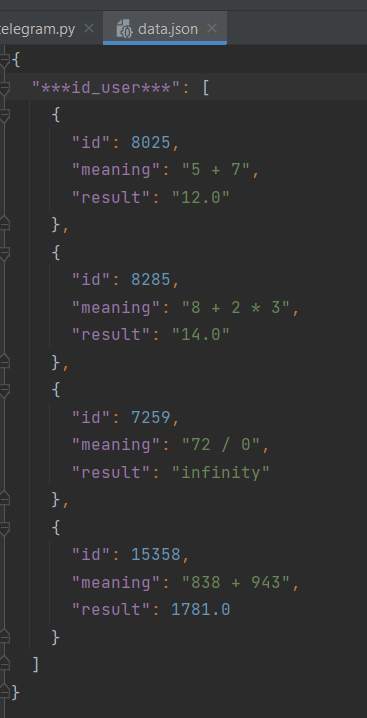
****

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

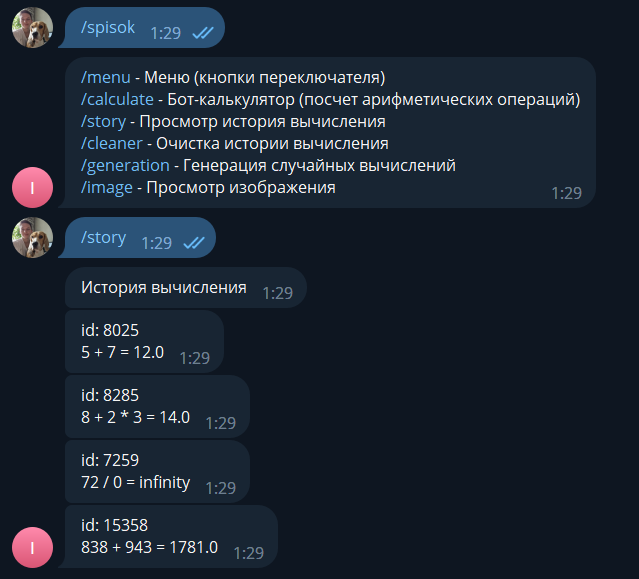
****

**Данные хранятся в data.json**

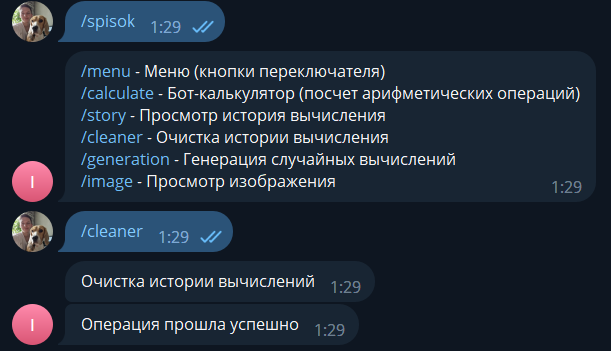
Вместо “\*\*\*id\_user\*\*\*” должны быть цифры индекса пользователя из телеграмма.

****

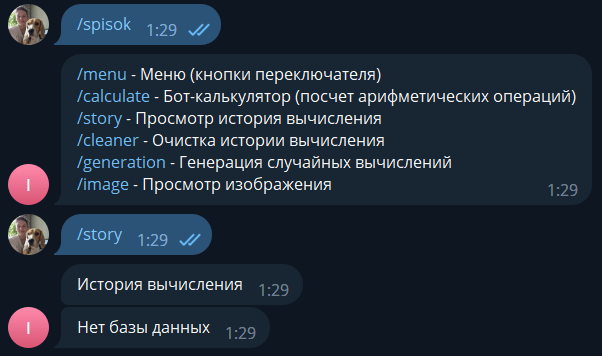
**Чтение и просмотр данные из data.json (при нажатии на ссылку «story»)**

****

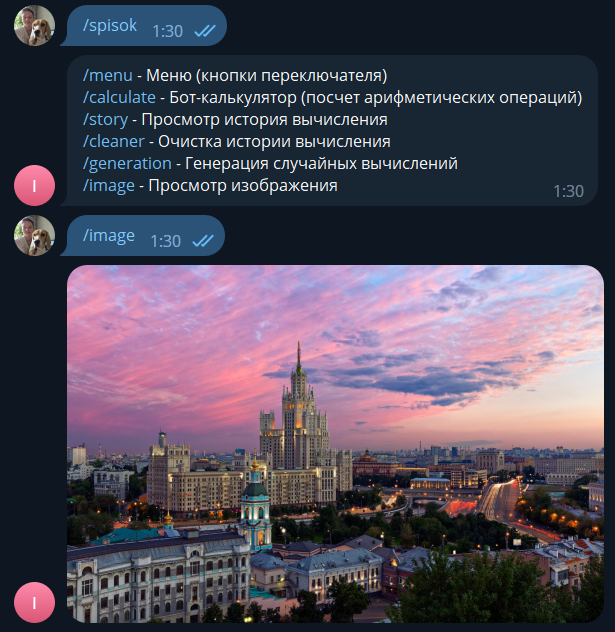
**Очистка истории вычисления (при нажатии на ссылку «/cleaner»)**

****

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

****

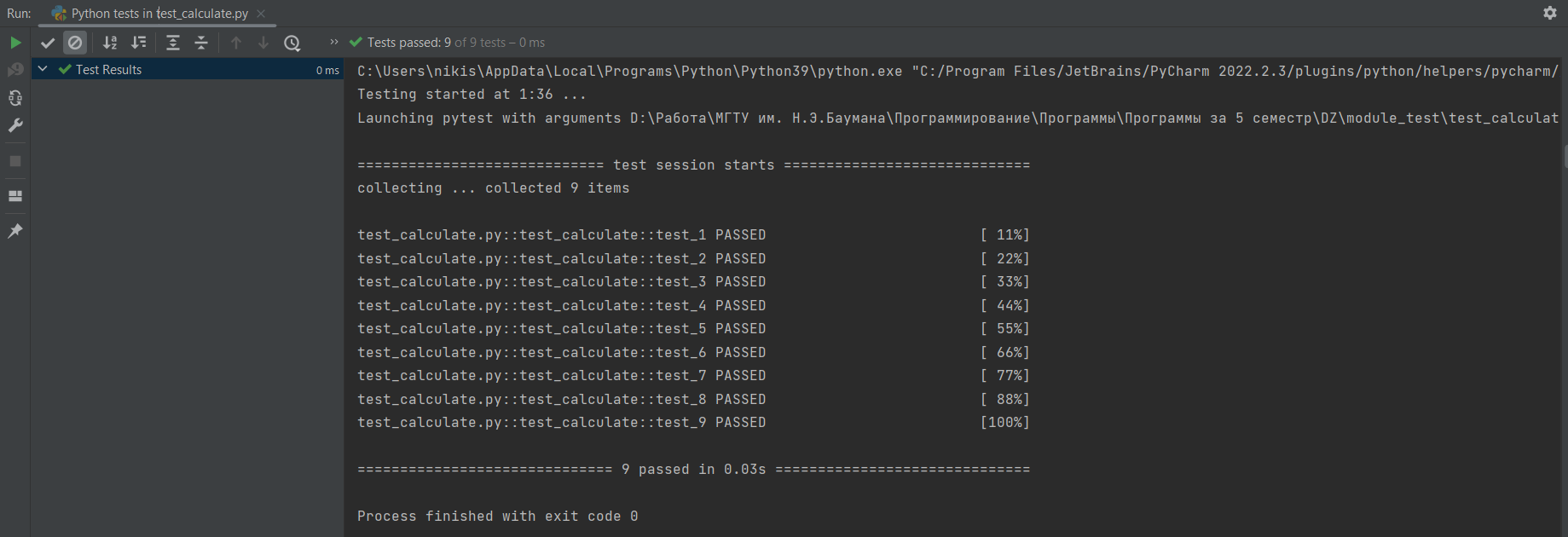
**Просмотр фотографии (при нажатии на ссылку «/image»)**

****

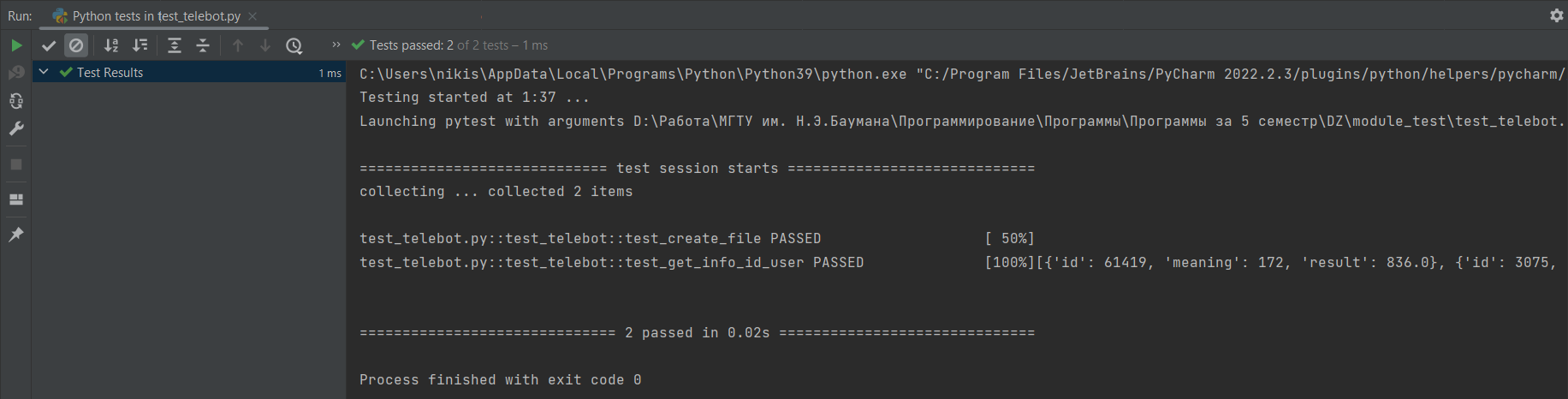
## Модульное тестирование

* 1. **Unittest**

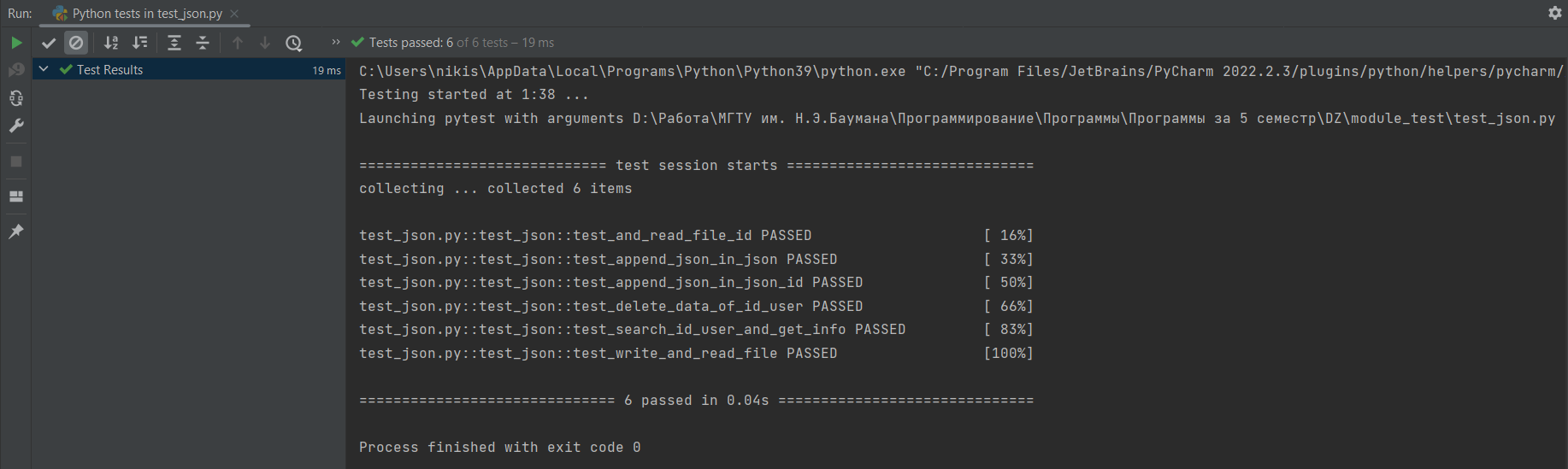
**Test\_calculate.py**

****

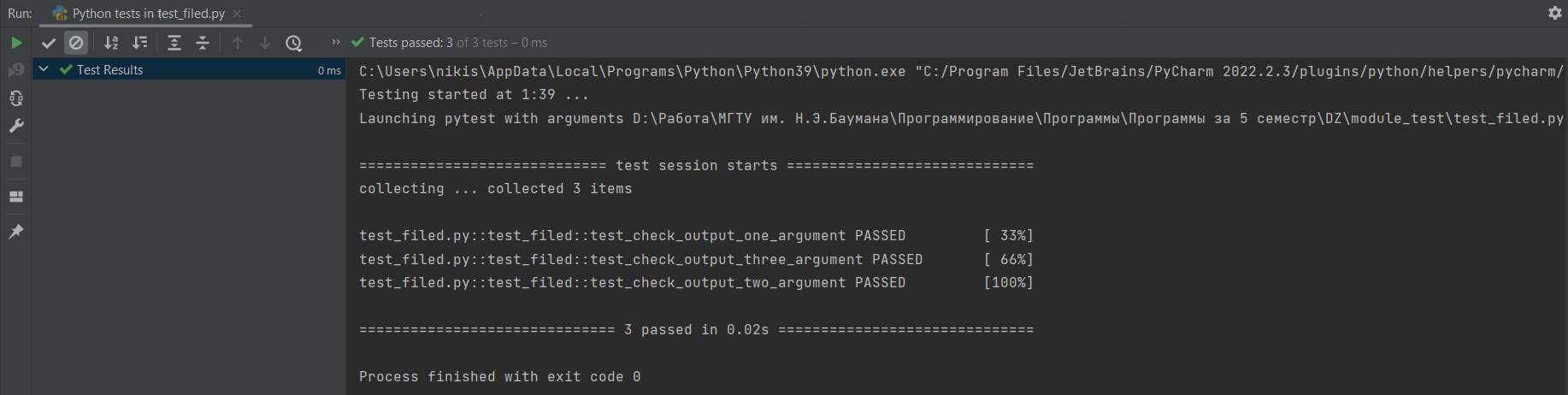
**Test\_telebot.py**

****

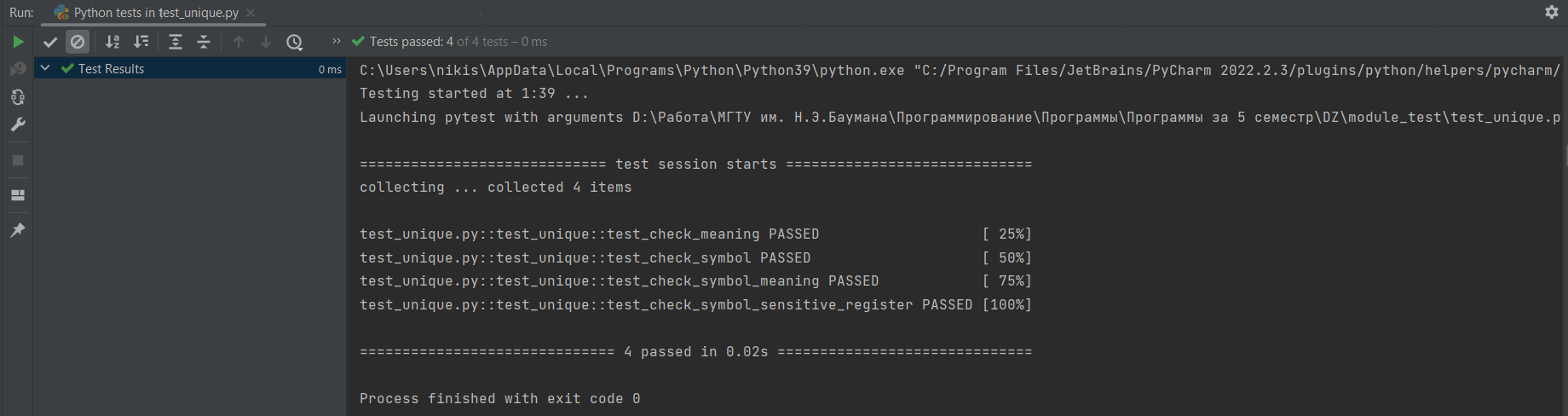
**Test\_json.py**

****

**Test\_filed.py**

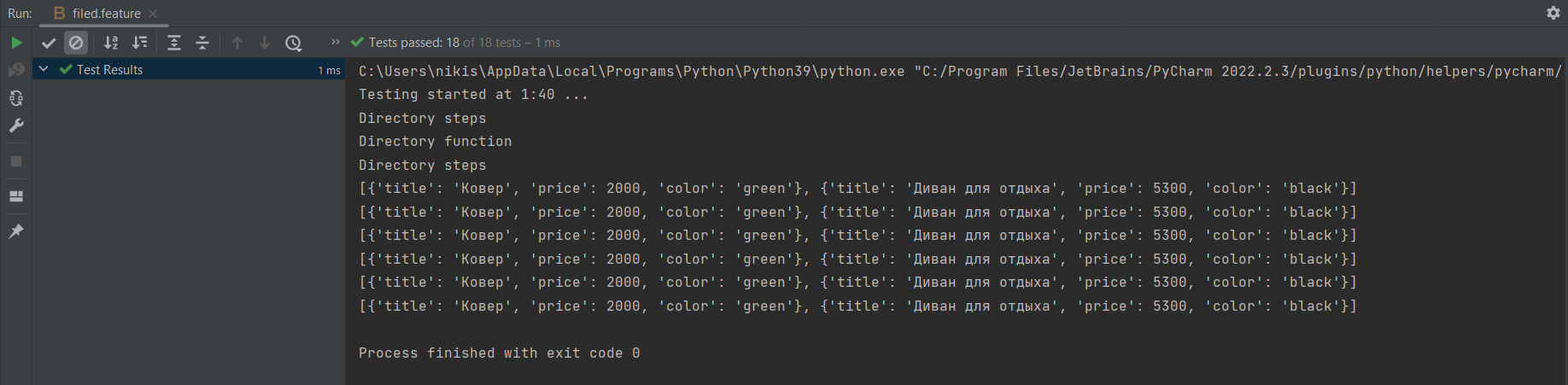
****

**Test\_unique.py**

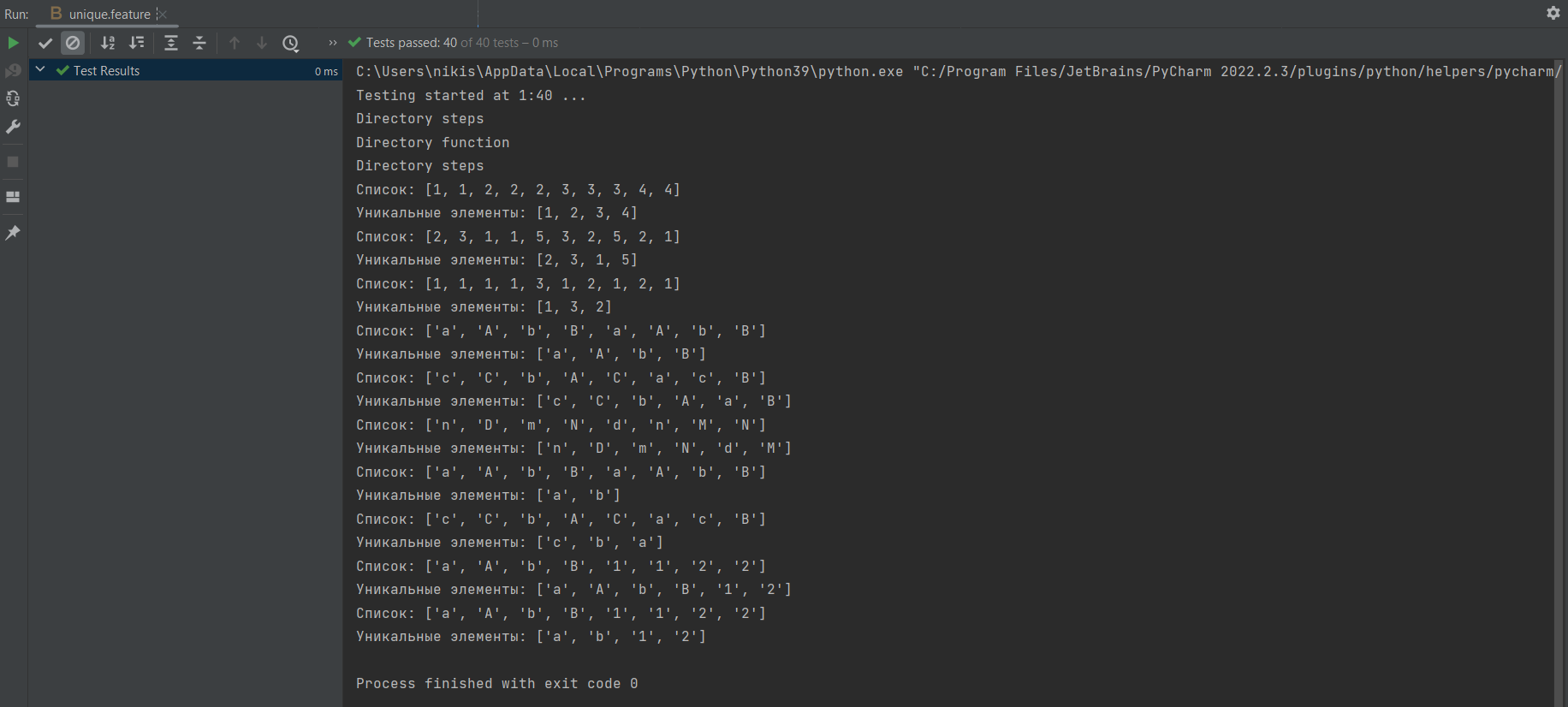
****

* 1. **Behave**

**Filed.feature**

****

**Unique.feature**

****