|  |  |  |
| --- | --- | --- |
| Защищено:  Папин А.В..    "\_\_" \_\_\_\_\_\_\_\_\_\_\_2022 г. |  | Демонстрация:  Папин А.В..    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

**Отчет по лабораторной работе №6 по курсу**

**базовые компоненты интернет-технологий (БКИТ)**

#### Домашнее задание

26

(количество листов)

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5Ц-54Б Папин Алексей | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Гапанюк Ю.Е. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

Москва, МГТУ - 2022

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ ОТЧЕТА**

[1. Цель лабораторной работы 2](#_Toc120647945)

[2. Описание задания. 2](#_Toc120647946)

[3. Листинг программы: 3](#_Toc120647947)

[3.1. config.py 3](#_Toc120647948)

[3.2. calculate\_arifmetic.py 3](#_Toc120647949)

[4.1. calculate\_bot.py 4](#_Toc120647950)

[4.2. json\_function.py 7](#_Toc120647951)

[4.3. work\_with\_calculate.py 7](#_Toc120647952)

[4.4. bmstu.jpg 8](#_Toc120647953)

[4.5. Unittest 9](#_Toc120647954)

[4.5.1.test\_calculate.py 9](#_Toc120647955)

[4.5.2.test\_telebot.py 9](#_Toc120647956)

[4.5.3.test\_json.py 10](#_Toc120647957)

[4.5.4.test\_filed.py (Unittest к 4 лабе) 12](#_Toc120647958)

[4.5.5.test\_unique.py (Unittest к 4 лабе) 13](#_Toc120647959)

[4.6. Behave 14](#_Toc120647960)

[4.6.1.check\_unique.feature 14](#_Toc120647961)

[4.6.2.check\_filed.feature 15](#_Toc120647962)

[4.7. Steps (for Behave) 16](#_Toc120647963)

[4.7.1.filed.py 16](#_Toc120647964)

[4.7.2.unique.py 17](#_Toc120647965)

[5. Результаты работы программы в Telegram 18](#_Toc120647966)

[5.1. Получение справочную информацию 18](#_Toc120647967)

[5.2. Основное меню переключателя 18](#_Toc120647968)

[5.3. Простейший калькулятор (при нажатии на кнопку «Посчитать») 18](#_Toc120647969)

[5.4. Данные хранятся в БД в формате JSON 19](#_Toc120647970)

[5.5. После несколько вычислений 19](#_Toc120647971)

[5.6. Обновленная БД 20](#_Toc120647972)

[5.7. Чтение и просмотр БД в Телеграме (после нажатии на кнопку Посмотреть историю вычисления) 21](#_Toc120647973)

[5.8. Просмотр фотографии (после нажатии на кнопку «Показать фото МГТУ им. Н.Э. Баумана») 22](#_Toc120647974)

[6. Модульное тестирование 23](#_Toc120647975)

[6.1. Unittest 23](#_Toc120647976)

[6.1.1.test\_unique.py 23](#_Toc120647977)

[6.1.2.test\_telebot.py 23](#_Toc120647978)

[6.1.3.test\_json.py 23](#_Toc120647979)

[6.1.4.test\_filed.py 24](#_Toc120647980)

[6.1.5.test\_calculate.py 24](#_Toc120647981)

[6.2. Behave 25](#_Toc120647982)

[6.2.1.check\_unique.feature 25](#_Toc120647983)

[6.2.2.check\_unique.feature 25](#_Toc120647984)

# **Цель лабораторной работы**

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).

# **Листинг программы:**

# **config.py**

token = ''

# **calculate\_arifmetic.py**

1. # Преобразование строкого типа в list  
   def delete\_space\_into\_list(string):  
    new\_str = []  
    str\_value = ''  
    for i in string:  
    if(i != ' '):  
    str\_value += i  
    else:  
    new\_str.append(str\_value)  
    str\_value = ''  
     
    new\_str.append(str\_value)  
     
    return new\_str  
     
     
   # Расстановка приоритета операции  
   def enumeration\_sign(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 = prioritet(count\_list)  
     
    return count\_list  
     
   # Поддержка функции по расстановку приоритета операции  
   def prioritet(list\_str):  
    new\_list = []  
    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 == '/'):  
    new\_list.append(i)  
    size -= 1  
    if('+' in list\_str or '-' in list\_str):  
    for i in list\_str:  
    if(i == '+' or i == '-'):  
    new\_list.append(i)  
    size -= 1  
     
    return new\_list  
     
   # Арифметические операции  
   def arifmetic(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]  
    except:  
    return result  
     
   def calculate(value):  
    new\_list = delete\_space\_into\_list(value)  
    list\_en = enumeration\_sign(new\_list)  
     
    for sgin in list\_en:  
    arifmetic(sgin, new\_list)  
     
    print(float(new\_list[0]))  
    return float(new\_list[0])

# **calculate\_bot.py**

import config  
import telebot  
from telebot import types  
import random  
  
from calculate\_arifmetic import calculate  
from calculate.work\_with\_calculate import get\_info  
from json\_function import merge\_data  
  
# Создание бота  
bot = telebot.TeleBot(config.token)  
  
HELP = '''  
/start - Меню переключателя  
/calculate - Калькуляторный бот, способный вычислять простейшие арифметические операции  
/get\_info - Просмотр историю вычисления с БД  
/photo - Просмотр фото МГТУ им. Н.Э. Баумана  
'''  
  
  
# Справочник  
@bot.message\_handler(commands=['help'])  
def start(message):  
 bot.send\_message(message.chat.id, HELP)  
  
  
@bot.message\_handler(commands=['start'])  
def start(message):  
 markup = types.InlineKeyboardMarkup(row\_width=1)  
 btn1 = types.InlineKeyboardButton(text="Посчитать", callback\_data='btn1')  
 btn2 = types.InlineKeyboardButton(text="Посмотреть историю вычисления", callback\_data='btn2')  
 btn3 = types.InlineKeyboardButton(text="Показать фото МГТУ им. Н.Э. Баумана", callback\_data='btn3')  
 markup.add(btn1, btn2, btn3)  
 bot.send\_message(message.chat.id,  
 text=f"Привет, {message.from\_user.first\_name}! Я тестовый бот, выберите действия",  
 reply\_markup=markup)  
  
  
# Функция переключателя  
@bot.callback\_query\_handler(func=lambda callback: callback.data)  
def check\_callback\_data(callback):  
 if (callback.data == "btn1"):  
 bot.send\_message(callback.message.chat.id, 'Калькулятор бот')  
 bot.send\_message(callback.message.chat.id, 'Напишите в чате вычисления')  
  
 @bot.message\_handler(content\_types=["text"])  
 def echo(message):  
 # Пользовательский идентификатор  
 user\_id = str(message.from\_user.id)  
  
 value = calculate(message.text)  
 bot.send\_message(message.chat.id, f'Решение: {value}')  
 data = {  
 user\_id: [  
 {"id": random.randint(0, 10000),  
 "value": str(message.text),  
 "result": str(value)}  
 ]  
 }  
 merge\_data(data, str(message.from\_user.id))  
  
 elif(callback.data == "btn2"):  
 bot.send\_message(callback.message.chat.id, 'История вычисления')  
 data = get\_info()  
 for i in data:  
 for j in data[i]:  
 id = j['id']  
 value = j['value']  
 result = j['result']  
 print\_info = f'id: {id}\n{value} = {result}\n\n'  
 bot.send\_message(callback.message.chat.id, print\_info)  
  
 elif(callback.data == "btn3"):  
 img = open('bmstu.jpg', 'rb')  
 bot.send\_photo(callback.message.chat.id, img)  
 else:  
 bot.send\_message(callback.chat.id, 'Нет такой команды. Введите /help')  
  
# Вычисления  
@bot.message\_handler(commands=['calculate'])  
def start\_calculate(message):  
 bot.send\_message(message.chat.id, 'Калькулятор бот')  
 bot.send\_message(message.chat.id, 'Напишите в чате вычисления')  
  
 # Пользовательский идентификатор  
 user\_id = str(message.from\_user.id)  
  
 @bot.message\_handler(content\_types=["text"])  
 def echo(message):  
 value = calculate(message.text)  
 bot.send\_message(message.chat.id, f'Решение: {value}')  
 data = {  
 user\_id: [  
 {"id": random.randint(0, 10000),  
 "value": str(message.text),  
 "result": str(value)}  
 ]  
 }  
 merge\_data(data, str(message.from\_user.id))  
  
  
# Просмотри история вычисления  
@bot.message\_handler(commands=['get\_info'])  
def start\_get\_info(message):  
 bot.send\_message(message.chat.id, 'История вычисления')  
  
 data = get\_info()  
 if (data == 'Файл отсутствует'):  
 bot.send\_message(message.chat.id, 'База данных отсутствует')  
 else:  
 for i in data:  
 for j in data[i]:  
 id = j['id']  
 value = j['value']  
 result = j['result']  
 print\_info = f'id: {id}\n{value} = {result}\n\n'  
 bot.send\_message(message.chat.id, print\_info)  
  
  
@bot.message\_handler(commands=['photo'])  
def url(message):  
 img = open('bmstu.jpg', 'rb')  
 bot.send\_photo(message.chat.id, img)  
  
  
  
bot.polling(none\_stop=True)

# **json\_function.py**

import json  
  
  
def write\_data(data, title='D:\Python\BKIT\calculate\data'):  
 with open(f"{title}.json", "w", encoding="utf-8") as file:  
 json.dump(data, file, indent=2, ensure\_ascii=False)  
  
  
def load\_data(title="D:\Python\BKIT\calculate\data"):  
 with open(f"{title}.json", "r") as file:  
 data = json.load(file)  
 return data  
  
  
def merge\_data(data\_json, id\_user='id\_user', title="D:\Python\BKIT\calculate\data"):  
 # Если файл существует 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]:  
 y = {  
 'id': info\_data['id'],  
 'value': info\_data['value'],  
 'result': info\_data['result']  
 }  
 temp.append(y)  
 write\_data(data)  
 # Если файл не существует  
 except:  
 write\_data(data\_json)

# **work\_with\_calculate.py**

import random  
  
from calculate.json\_function import write\_data, load\_data, merge\_data  
from calculate.calculate\_arifmetic import calculate  
  
  
def generate\_value(id\_user='id\_user'):  
 arifmetic = ['+', '-', '/', '\*']  
  
 af = arifmetic[random.randint(0, 3)]  
 gen\_id = random.randint(0, 100000)  
 v1 = random.randint(0, 1000)  
 v2 = random.randint(0, 1000)  
 result = calculate(str(v1) + ' ' + str(af) + ' ' + str(v2))  
  
 data = {  
 str(id\_user): [  
 {  
 "id": gen\_id,  
 "value": v1,  
 "result": result  
 }  
 ]  
 }  
  
 merge\_data(data, id\_user)  
  
def get\_info():  
 try:  
 data = load\_data()  
 return data  
 except:  
 return 'Файл отсутствует'

# **bmstu.jpg**



# **Unittest**

# **test\_calculate.py**

import unittest  
  
from calculate.calculate\_arifmetic import calculate  
  
class test\_calculate(unittest.TestCase):  
  
 # Проверка на работу  
 def test\_1(self):  
 self.assertEqual(calculate('10'), 10.0)  
  
 def test\_2(self):  
 self.assertEqual(calculate('10 + 10'), 20.0)  
  
 def test\_3(self):  
 self.assertEqual(calculate('2 + 3 \* 2'), 8.0)  
  
 def test\_4(self):  
 self.assertEqual(calculate('2 + 3 + 2'), 7.0)  
  
 def test\_5(self):  
 self.assertEqual(calculate('2 + 3 - 2'), 3.0)  
  
 def test\_6(self):  
 self.assertEqual(calculate('5 / 2 \* 2'), 5.0)  
  
 def test\_7(self):  
 self.assertEqual(calculate('100 - 10 + 100'), 190.0)

# **test\_telebot.py**

import unittest  
import os.path  
  
from calculate.work\_with\_calculate import generate\_value  
  
  
class test\_telebot(unittest.TestCase):  
  
 # Проверка создания файла и наличия файла  
 def test\_create\_file\_json(self):  
 message\_from\_user\_id = 369350478  
  
 generate\_value(str(message\_from\_user\_id))  
  
 self.assertEqual(  
 os.path.exists('D:\Python\BKIT\calculate\data.json'),  
 True  
 )

# **test\_json.py**

import unittest  
  
from calculate.json\_function import load\_data, write\_data, merge\_data  
  
data\_json = {  
 "id\_user": [  
 {  
 "id": 12425,  
 "value": '30 + 40',  
 "result": '70'  
 }  
 ]  
}  
  
data\_json\_big = {  
 "id\_user": [  
 {  
 "id": 52478,  
 "value": '10 + 10',  
 "result": '20'  
 },  
 {  
 "id": 5437,  
 "value": '10 + 10',  
 "result": '20'  
 },  
 {  
 "id": 69823,  
 "value": '10 + 10',  
 "result": '20'  
 },  
 {  
 "id": 24537,  
 "value": '10 + 10',  
 "result": '20'  
 },  
 {  
 "id": 786,  
 "value": '10 + 10',  
 "result": '20'  
 }  
 ]  
}  
  
data\_json1 = {  
 "id\_user": [  
 {  
 "id": 324,  
 "value": '50 + 50',  
 "result": '100'  
 }  
 ]  
}  
  
data\_json\_with\_id = {  
 "369350471": [  
 {  
 "id": 12425,  
 "value": '30 + 40',  
 "result": '70'  
 }  
 ]  
}  
  
data\_json\_with\_id\_1 = {  
 "369350471": [  
 {  
 "id": 78678,  
 "value": '70 + 40',  
 "result": '110'  
 }  
 ]  
}  
  
  
class test\_json(unittest.TestCase):  
  
 # Проверка на присутствия файла  
 def test\_write\_and\_read\_file(self):  
 # Создаем файл с данным  
 write\_data(data\_json)  
  
 # Проверяем на наличие и сходимости  
 self.assertEqual(  
 load\_data(),  
 {'id\_user': [{'id': 12425, 'result': '70', 'value': '30 + 40'}]}  
 )  
  
 # Проверка на добавлении json дата  
 def test\_append\_json\_in\_json(self):  
 # Создаем файл с данным  
 write\_data(data\_json)  
  
 # Изменяем файл - добавление новые данных  
 merge\_data(data\_json1)  
  
 # Проверяем на наличие и сходимости  
 self.assertEqual(  
 load\_data(),  
 {'id\_user': [  
 {'id': 12425, 'result': '70', 'value': '30 + 40'},  
 {'id': 324, 'result': '100', 'value': '50 + 50'}  
 ]})  
  
 # Проверка на добавлении json дата с идентификатором пользователя  
 def test\_and\_read\_file\_with\_id(self):  
 # Создаем файл с данным  
 write\_data(data\_json\_with\_id)  
  
 # Проверяем на наличие и сходимости  
 self.assertEqual(  
 load\_data(),  
 {'369350471': [{'id': 12425, 'result': '70', 'value': '30 + 40'}]}  
 )  
  
 # Проверка на добавлении json дата с идентификатором пользователя  
 def test\_append\_json\_in\_json\_with\_id(self):  
 # Создаем файл с данным  
 write\_data(data\_json\_with\_id)  
  
 # Изменяем файл - добавление новые данных  
 merge\_data(data\_json\_with\_id\_1, str(369350471))  
  
 # Проверяем на наличие и сходимости  
 self.assertEqual(  
 load\_data(),  
 {'369350471': [  
 {'id': 12425, 'result': '70', 'value': '30 + 40'},  
 {'id': 78678, 'result': '110', 'value': '70 + 40'}  
 ]})

# **test\_filed.py (Unittest к 4 лабе)**

# Подключаем библиотеку unitetest для тестирования  
import unittest  
  
'''  
assertEqual(self, first, second)  
first - передаваемое значение  
second - полученное значение (в тело функции должен быть return, если вы там не оставили, тогда прописать здесь как None)  
если передаваемое значение совпадает с полученным значением, то тест пройден успешно  
'''  
  
from function.filed import field, goods  
  
  
class test\_filed(unittest.TestCase):  
  
 # Проверка вывода с 1 аргумента  
 def test\_pass\_one\_argv(self):  
 self.assertEqual(  
 field(goods, 'title'),  
 [  
 {'title': 'Ковер'},  
 {'title': 'Диван для отдыха'}  
 ]  
 )  
  
 # Проверка вывода с 2 аргумента  
 def test\_pass\_two\_argv(self):  
 self.assertEqual(  
 field(goods, 'title color'),  
 [  
 {'color': 'green', 'title': 'Ковер'},  
 {'color': 'black', 'title': 'Диван для отдыха'}  
 ]  
 )  
  
 # Проверка вывода с 3 аргумента  
 def test\_pass\_three\_argv(self):  
 self.assertEqual(  
 field(goods, 'title color price'),  
 [  
 {'color': 'green', 'price': 2000, 'title': 'Ковер'},  
 {'color': 'black', 'price': 5300, 'title': 'Диван для отдыха'}  
 ]  
 )

# **test\_unique.py (Unittest к 4 лабе)**

# Подключаем библиотеку unitetest для тестирования  
import unittest  
  
'''  
assertEqual(self, first, second)  
first - передаваемое значение  
second - полученное значение (в тело функции должен быть return, если вы там не оставили, тогда прописать здесь как None)  
если передаваемое значение совпадает с полученным значением, то тест пройден успешно  
'''  
  
from function.unique import Unique  
  
class test\_unique(unittest.TestCase):  
 # Проверка на чисел  
 def test\_value(self):  
 # Дан список с числами  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 # Получаем уникальные значения и сохраним его в переменной  
 arr\_unique = Unique(data).arr  
 # Проверяем  
 self.assertEqual(  
 arr\_unique,  
 [1, 2]  
 )  
  
 # Проверка на буквы  
 def test\_letters(self):  
 # Дан список с числами  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 # Получаем уникальные значения и сохраним его в переменной  
 arr\_unique = Unique(data).arr  
 # Проверяем  
 self.assertEqual(  
 arr\_unique,  
 ['a', 'A', 'b', 'B']  
 )  
  
 # Проверка на буквы без чувствительного регистра  
 def test\_letters\_ignore\_case(self):  
 # Дан список с числами  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 # Получаем уникальные значения и сохраним его в переменной  
 arr\_unique = Unique(data, ignore\_case = True).arr  
 # Проверяем  
 self.assertEqual(  
 arr\_unique,  
 ['a', 'b']  
 )  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

# **Behave**

# **check\_unique.feature**

Feature: Calculating and getting unique values  
  
 *# Уникальные значения числового типа  
 # If <CASE> is 1 then is True* Scenario Outline: We get unique values from the list of the contained number  
 Given I have a class of unique values  
 And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<unique>** Examples:  
 | **list** | **unique** | **CASE** |  
 | **[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]** | **[1, 2]** | **0** |  
 | **[1, 3, 1, 1, 1, 3, 2, 2, 2, 2]** | **[1, 3, 2]** | **0** |  
  
  
 *# Уникальные значения символьного типа* Scenario Outline: We get unique values from the list of the contained char  
 Given I have a class of unique values  
 And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<unique>** Examples:  
 | **list** | **unique** | **CASE** |  
 | **['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']** | **['a', 'A', 'b', 'B']** | **0** |  
 | **['a', 'C', 'b', 'B', 'c', 'A', 'b', 'B']** | **['a', 'C', 'b', 'B', 'c', 'A']** | **0** |  
  
  
 *# Уникальные значения символьного типа без чувствительного регистра* Scenario Outline: We get unique values from the list of the contained char ignore\_case  
 Given I have a class of unique values  
 And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<unique>** Examples:  
 | **list** | **unique** | **CASE** |  
 | **['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']** | **['a', 'b']** | **1** |  
 | **['a', 'C', 'b', 'B', 'c', 'A', 'b', 'B']** | **['a', 'c', 'b']** | **1** |  
  
  
 *# Уникальные значения смешанного типа* Scenario Outline: We get unique values from the list of the contained all type  
 Given I have a class of unique values  
 And Getting the list: **<list>** When Finding unique values, case: **<CASE>** Then Output unique values: **<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** |

# **check\_filed.feature**

Feature: Checking the output of an argument from the goods dictionaries  
  
  
 *# Проверка вывода с 1 аргумента* Scenario Outline: Checking the output with 1 argument  
 Given I have a dictionary goods  
 When We enter **<arguments>** to get the desired values  
 Then Output to the **<check\_result>** Examples:  
 | **arguments** | **check\_result** |  
 | **title** | **[{'title': 'Ковер'},{'title': 'Диван для отдыха'}]** |  
 | **color** | **[{'color': 'green'},{'color': 'black'}]** |  
 | **price** | **[{'price': 2000},{'price': 5300}]** |  
  
  
 *# Проверка вывода с 2 аргумента* Scenario Outline: Checking the output with 2 argument  
 Given I have a dictionary goods  
 When We enter **<arguments>** to get the desired values  
 Then Output to the **<check\_result>** Examples:  
 | **arguments** | **check\_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 с 3 аргумента* Scenario Outline: Checking the output with 3 argument  
 Given I have a dictionary goods  
 When We enter **<arguments>** to get the desired values  
 Then Output to the **<check\_result>** Examples:  
 | **arguments** | **check\_result** |  
 | **title color price** | **[{'color': 'green', 'price': 2000, 'title': 'Ковер'}, {'color': 'black', 'price': 5300, 'title': 'Диван для отдыха'}]** |

# **Steps (for Behave)**

# **filed.py**

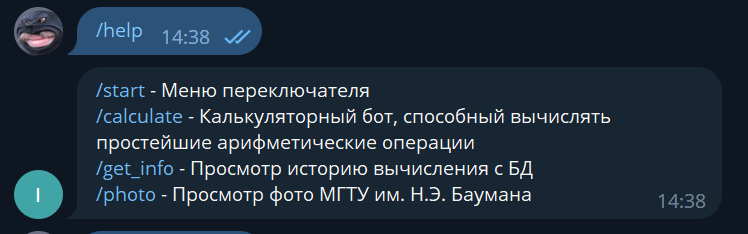
from behave import Given, When, Then  
from function.filed import field, goods  
import ast  
  
@Given('I have a dictionary goods')  
def step\_impl(context):  
 context.data\_dictonary = goods  
 test = context.data\_dictonary  
 print(test)  
  
  
@When("We enter {arguments} to get the desired values")  
def given\_increment(context, arguments):  
 context.results = field(context.data\_dictonary, arguments)  
  
  
@Then("Output to the {check\_result}")  
def then\_results(context, check\_result):  
 assert context.results == ast.literal\_eval(check\_result)

# **unique.py**

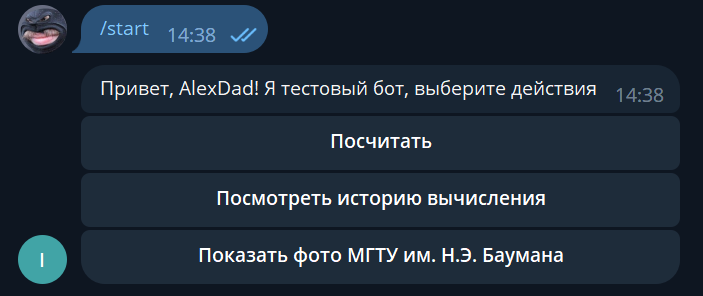
from behave import Given, When, Then  
from function.unique import Unique  
import ast  
  
  
@Given('I have a class of unique values')  
def step\_impl(context):  
 pass  
  
  
@Given("Getting the list: {LIST}")  
def given\_increment(context, LIST):  
 context.LIST = list(ast.literal\_eval(LIST))  
 print(f'Список: {LIST}')  
  
  
@When("Finding unique values, case: {CASE}")  
def given\_increment(context, CASE):  
 check = bool(int(CASE))  
 if (check == True):  
 unique\_list = Unique(context.LIST, ignore\_case=check)  
 else:  
 unique\_list = Unique(context.LIST)  
  
 context.results = unique\_list  
 # print(f'Уникальные значения: {unique\_list}')  
  
  
@Then("Output unique values: {UNIQUE}")  
def then\_results(context, UNIQUE):  
 assert context.results.arr == ast.literal\_eval(UNIQUE)  
 print(f'Уникальные значения: {context.results.arr}')

# **Результаты работы программы в Telegram**

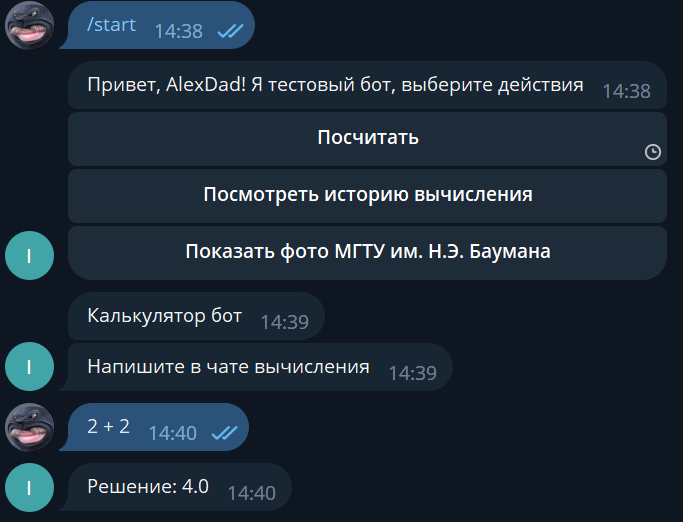
# **Получение справочную информацию**



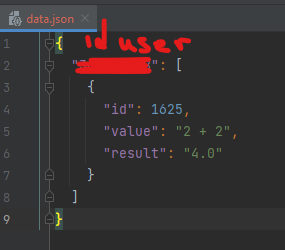
# **Основное меню переключателя**



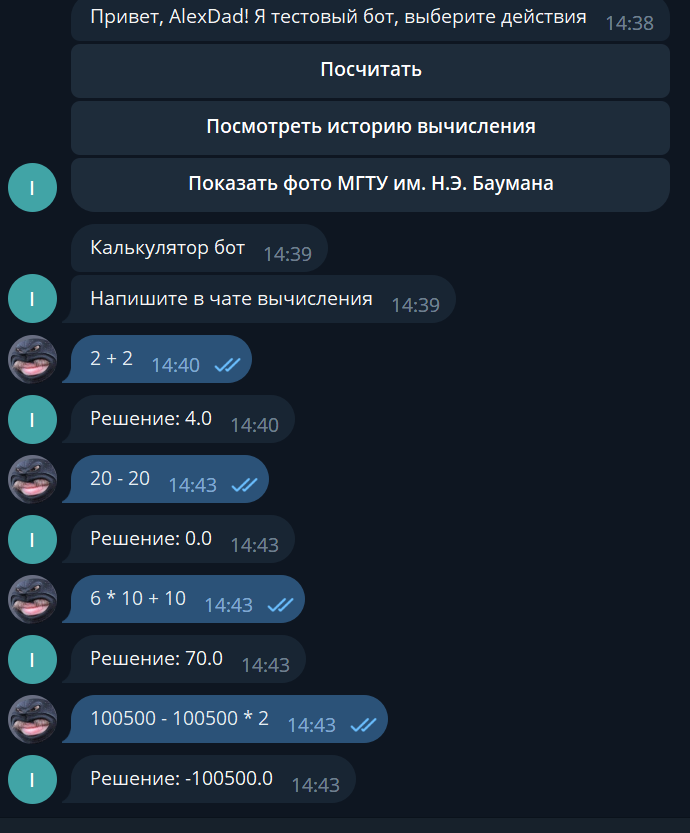
# **Простейший калькулятор (при нажатии на кнопку «Посчитать»)**



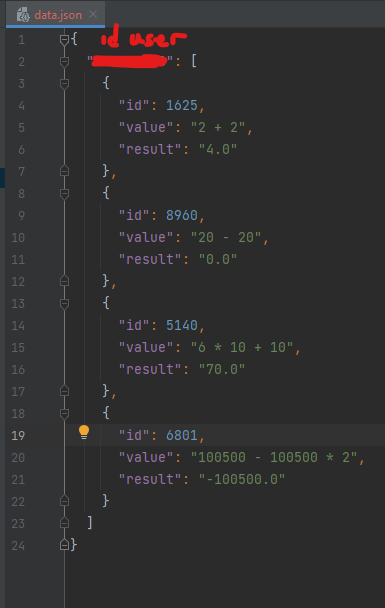
# **Данные хранятся в БД в формате JSON**



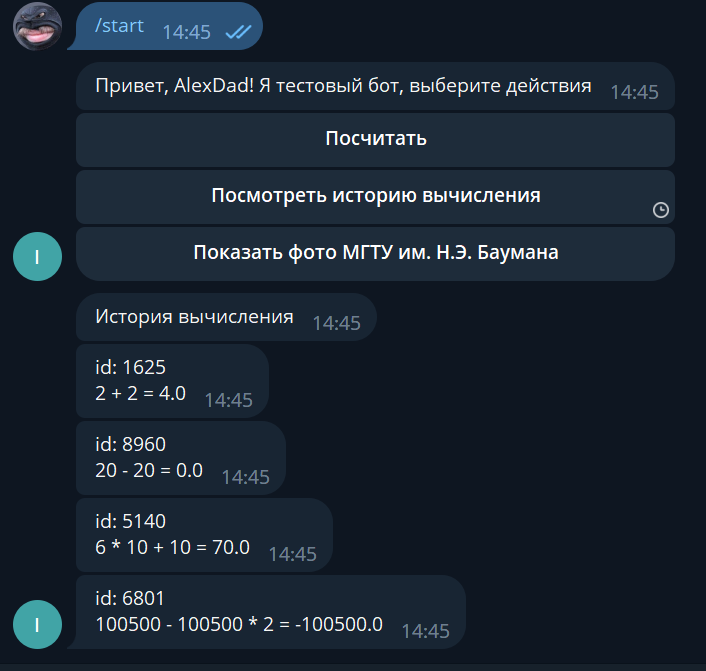
# **После несколько вычислений**



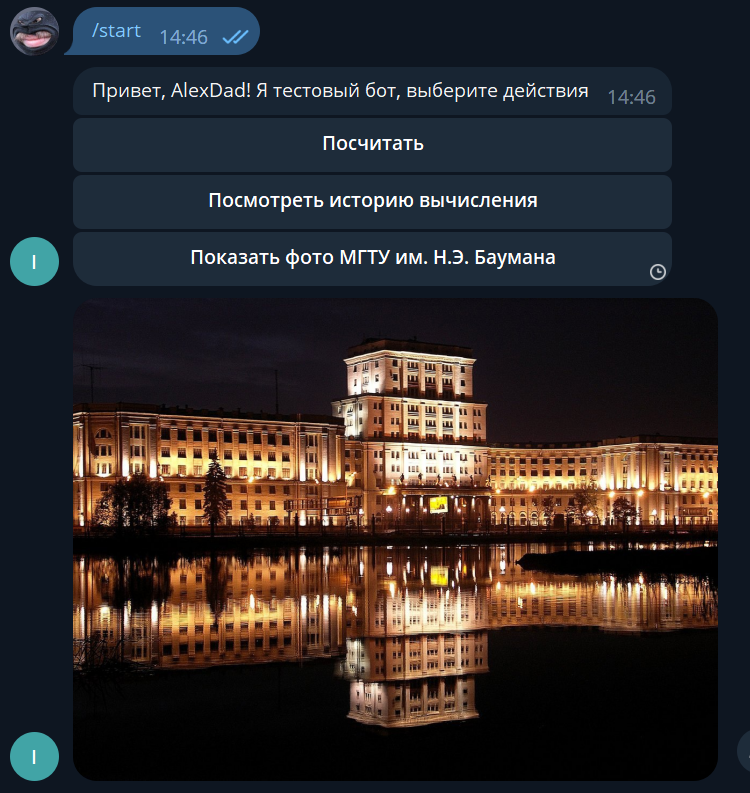
# **Обновленная БД**



# **Чтение и просмотр БД в Телеграме (после нажатии на кнопку Посмотреть историю вычисления)**



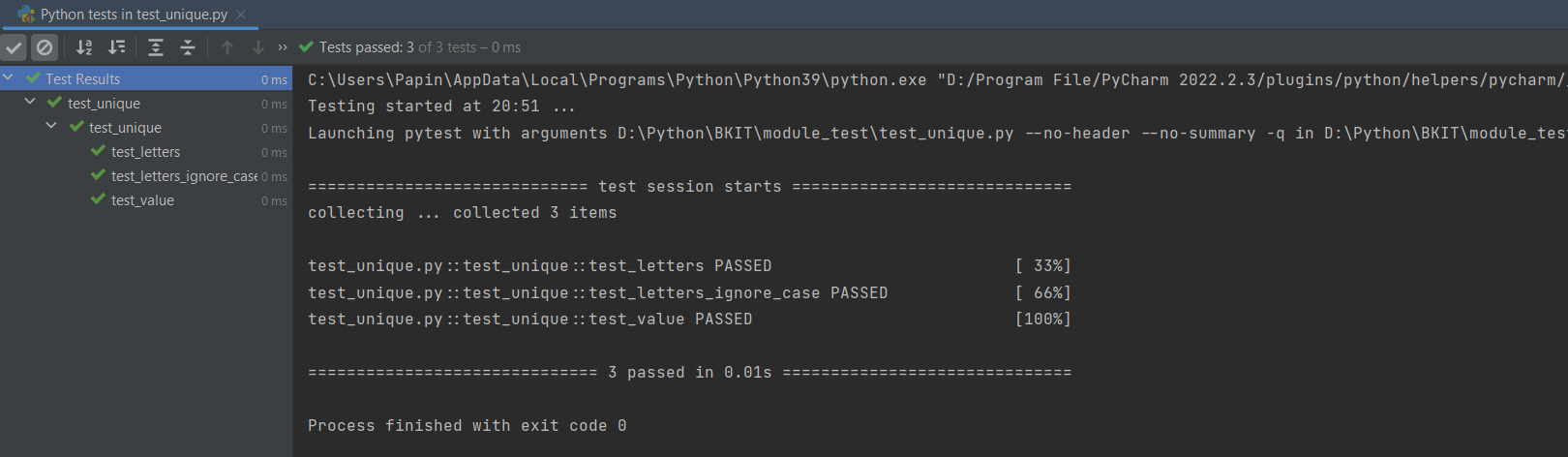
# **Просмотр фотографии (после нажатии на кнопку «Показать фото МГТУ им. Н.Э. Баумана»)**



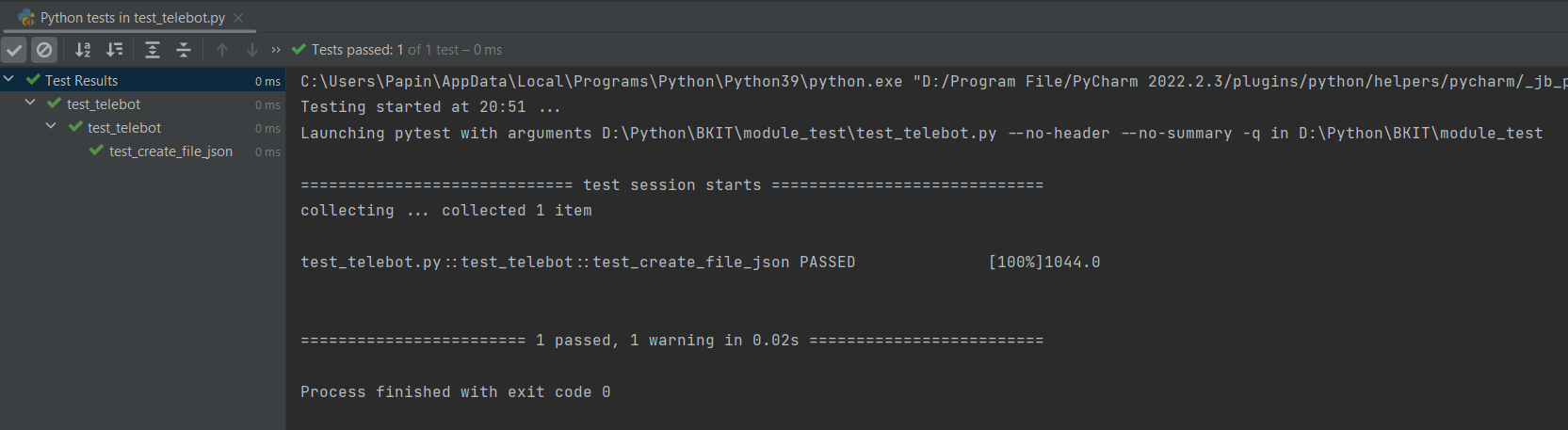
# **Модульное тестирование**

# **Unittest**

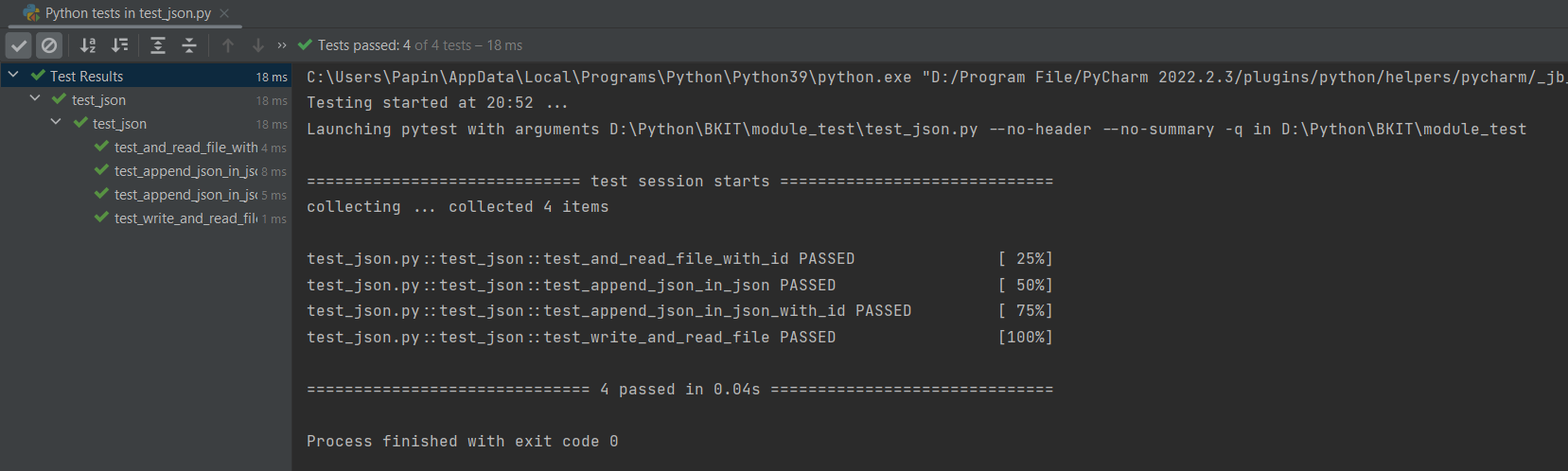
# **test\_unique.py**



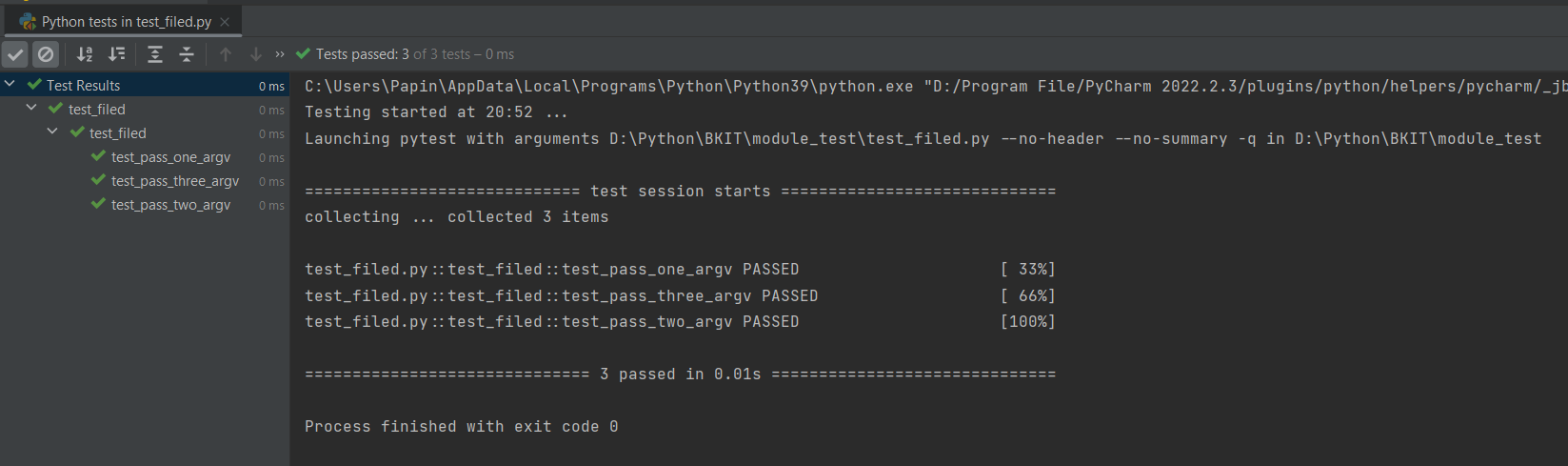
# **test\_telebot.py**



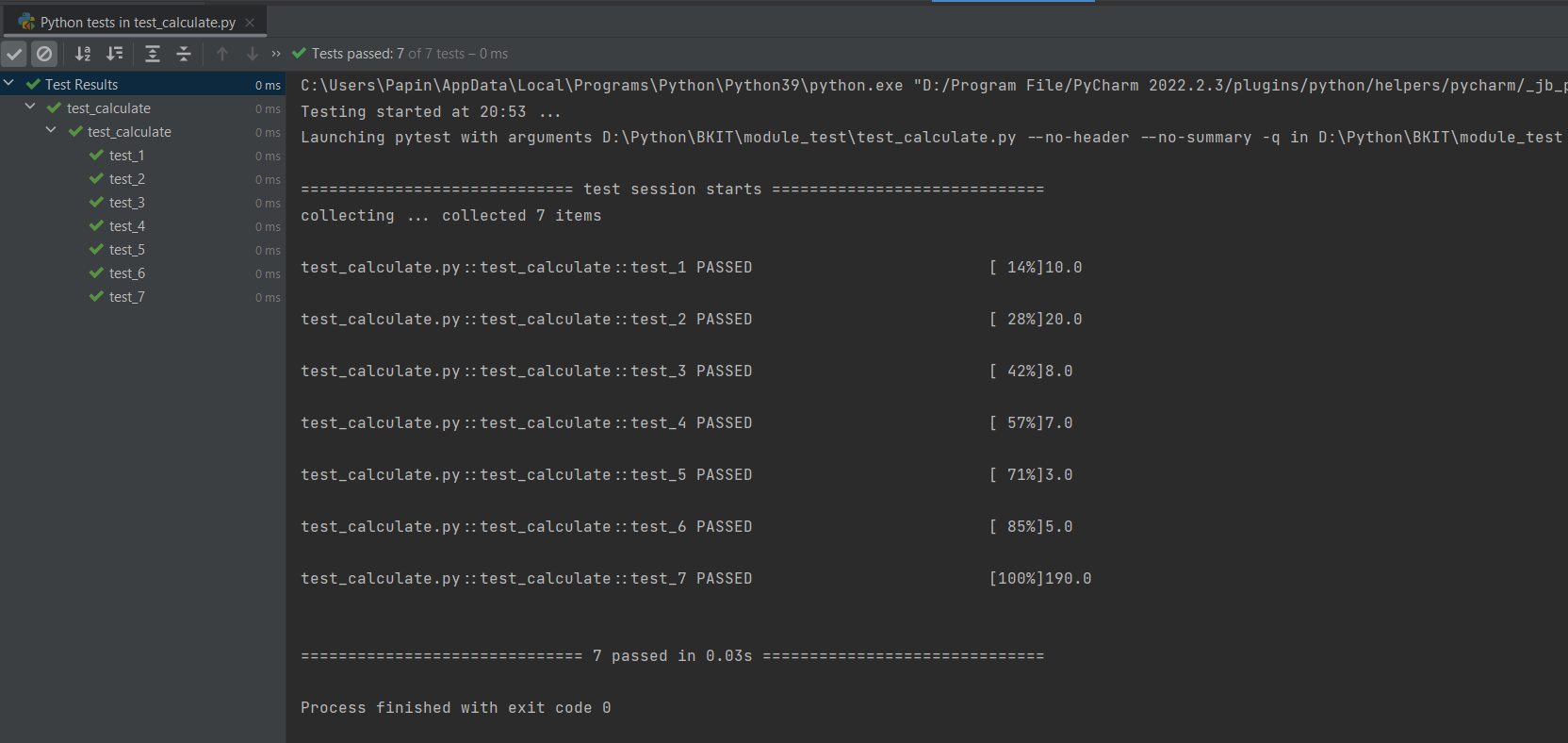
# **test\_json.py**



# **test\_filed.py**

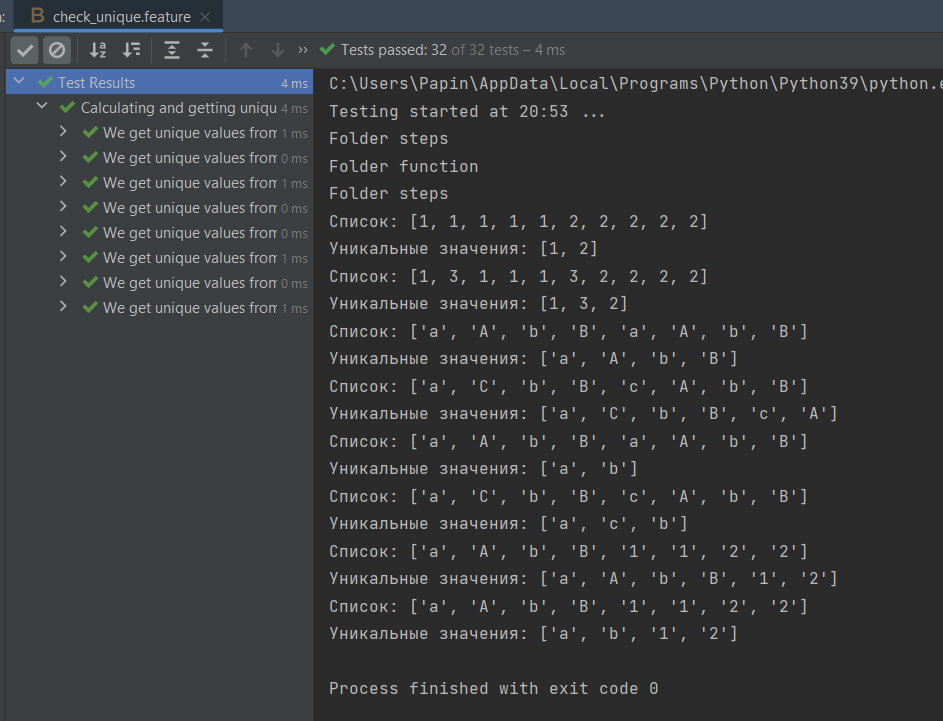


# **test\_calculate.py**



# **Behave**

# **check\_unique.feature**



# **check\_unique.feature**

