**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Бондаренко Денис Константинович |  | Гапанюк Юрий Евгеньевич |
|  |  |  |
| Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
|  |  |  |

Москва, 2021 г.

**Домашнее задание**

**Описание задания**

1. Модифицируйте код лабораторной работы №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**

**dbworker.py**

from vedis import Vedis  
import config  
  
  
# Чтение значения  
def get(key):  
 with Vedis(config.db\_file) as db:  
 try:  
 return db[key].decode()  
 except KeyError:  
 # в случае ошибки значение по умолчанию - начало диалога  
 return config.States.STATE\_START.value  
  
  
# Запись значения  
def set(key, value):  
 with Vedis(config.db\_file) as db:  
 try:  
 db[key] = value  
 return True  
 except:  
 return False  
  
  
# Создание ключа для записи и чтения  
def make\_key(chatid, keyid):  
 res = str(chatid) + '\_\_' + str(keyid)  
 return res

**config.py**

from enum import Enum  
  
# Токент бота  
TOKEN = "5084251495:AAGvFx\_LRatbRHFGi68RI-dHZCyPb0QtwUs"  
  
# Файл базы данных Vedis  
db\_file = "db.vdb"  
  
# Матрицы для знаков зодиака  
love\_comp = [[True, False, False, False],  
 [False, True, False, True],  
 [False, False, False, True],  
 [False, True, False, False]]  
  
work\_comp = [[True, False, True, False],  
 [True, True, True, False],  
 [False, False, True, True],  
 [True, False, False, False]]  
  
# Словарь для знаков зодиака  
Zodiac = {  
 0: "Овен",  
 1: "Телец",  
 2: "Близнецы",  
 3: "Рак"  
}  
  
  
# Функция при полученных знаках и типе отношения  
def conclusion(s1, s2, rltn):  
 n1 = int(s1)  
 n2 = int(s2)  
 res = True  
 if rltn == 'Любовь':  
 res = love\_comp[n1][n2]  
 elif rltn == 'Работа':  
 res = work\_comp[n1][n2]  
 return res  
  
  
# Ключ записи в БД для текущего состояния  
CURRENT\_STATE = "CURRENT\_STATE"  
  
  
# Состояния автомата  
class States(Enum):  
 STATE\_START = "STATE\_START" # Начало нового диалога  
 STATE\_FIRST\_NUM = "STATE\_FIRST\_NUM"  
 STATE\_SECOND\_NUM = "STATE\_SECOND\_NUM"  
 STATE\_OPERATION = "STATE\_OPERATION"

**bot2.py**

import telebot  
from telebot import types  
import config  
import dbworker  
  
# Создание бота  
bot = telebot.TeleBot(config.TOKEN)  
  
  
# Начало диалога  
@bot.message\_handler(commands=['start'])  
def cmd\_start(message):  
 bot.send\_message(message.chat.id, 'Я умею проверять представителей знаков Зодиака на совместимость!\n'  
 '0 - Овен\n'  
 '1 - Телец\n'  
 '2 - Близнецы\n'  
 '3 - Рак')  
 dbworker.set(dbworker.make\_key(message.chat.id, config.CURRENT\_STATE), config.States.STATE\_FIRST\_NUM.value)  
 bot.send\_message(message.chat.id, 'Введите номер первого знака Зодиака')  
  
  
# По команде /reset будем сбрасывать состояния, возвращаясь к началу диалога  
@bot.message\_handler(commands=['reset'])  
def cmd\_reset(message):  
 bot.send\_message(message.chat.id, 'Сбрасываем результаты предыдущих вводов.')  
 dbworker.set(dbworker.make\_key(message.chat.id, config.CURRENT\_STATE), config.States.STATE\_FIRST\_NUM.value)  
 bot.send\_message(message.chat.id, 'Введите номер первого знака Зодиака')  
  
  
# Обработка первого числа (знака Зодиака)  
@bot.message\_handler(func=lambda message: dbworker.get(  
 dbworker.make\_key(message.chat.id, config.CURRENT\_STATE)) == config.States.STATE\_FIRST\_NUM.value)  
def first\_num(message):  
 text = message.text  
 if not (text.isdigit() and int(text)>=0 and int(text)<=config.Zodiac.\_\_len\_\_()-1):  
 # Состояние не изменяется, выводится сообщение об ошибке  
 bot.send\_message(message.chat.id, 'Пожалуйста введите подходящий номер!')  
 return  
 else:  
 bot.send\_message(message.chat.id, f'Ваш первый знак Зодиака - {config.Zodiac[int(text)]}')  
 # Меняем текущее состояние  
 dbworker.set(dbworker.make\_key(message.chat.id, config.CURRENT\_STATE), config.States.STATE\_SECOND\_NUM.value)  
 # Сохраняем первое число (знак Зодиака)  
 dbworker.set(dbworker.make\_key(message.chat.id, config.States.STATE\_FIRST\_NUM.value), text)  
 bot.send\_message(message.chat.id, 'Введите номер второго знака Зодиака')  
  
  
# Обработка второго числа (знака Зодиака)  
@bot.message\_handler(func=lambda message: dbworker.get(  
 dbworker.make\_key(message.chat.id, config.CURRENT\_STATE)) == config.States.STATE\_SECOND\_NUM.value)  
def second\_num(message):  
 text = message.text  
 if not (text.isdigit() and int(text)>=0 and int(text)<=config.Zodiac.\_\_len\_\_()-1):  
 # Состояние не изменяется, выводится сообщение об ошибке  
 bot.send\_message(message.chat.id, 'Пожалуйста введите подходящий номер!')  
 return  
 else:  
 bot.send\_message(message.chat.id, f'Ваш второй знак Зодиака - {config.Zodiac[int(text)]}')  
 # Меняем текущее состояние  
 dbworker.set(dbworker.make\_key(message.chat.id, config.CURRENT\_STATE), config.States.STATE\_OPERATION.value)  
 # Сохраняем второе число (знак Зодиака)  
 dbworker.set(dbworker.make\_key(message.chat.id, config.States.STATE\_SECOND\_NUM.value), text)  
 markup = types.ReplyKeyboardMarkup(row\_width=2)  
 itembtn1 = types.KeyboardButton('Любовь')  
 itembtn2 = types.KeyboardButton('Работа')  
 markup.add(itembtn1, itembtn2)  
 bot.send\_message(message.chat.id, 'Выберите пожалуйста тип отношений', reply\_markup=markup)  
  
  
# Выбор типа отношений  
@bot.message\_handler(func=lambda message: dbworker.get(  
 dbworker.make\_key(message.chat.id, config.CURRENT\_STATE)) == config.States.STATE\_OPERATION.value)  
def relation(message):  
 # Текущий тип отношения  
 rltn = message.text  
 # Читаем номера знаков Зодиака из базы данных  
 s1 = dbworker.get(dbworker.make\_key(message.chat.id, config.States.STATE\_FIRST\_NUM.value))  
 s2 = dbworker.get(dbworker.make\_key(message.chat.id, config.States.STATE\_SECOND\_NUM.value))  
 # Выводим результат  
 markup = types.ReplyKeyboardRemove(selective=False)  
  
 if config.conclusion(s1, s2, rltn):  
 bot.send\_message(message.chat.id, f'Результат: {config.Zodiac[int(s1)]} и {config.Zodiac[int(s2)]} {"cовместимы"}'  
 f' при типе отношений: {rltn}', reply\_markup=markup)  
 else:  
 bot.send\_message(message.chat.id, f'Результат: {config.Zodiac[int(s1)]} и {config.Zodiac[int(s2)]} {"неcовместимы"}'  
 f' при типе отношений: {rltn}', reply\_markup=markup)  
 # Меняем текущее состояние  
 dbworker.set(dbworker.make\_key(message.chat.id, config.CURRENT\_STATE), config.States.STATE\_FIRST\_NUM.value)  
 # Выводим сообщение  
 bot.send\_message(message.chat.id, 'Введите номер первого знака Зодиака')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 bot.infinity\_polling()

**bot\_test.py**

import unittest  
from bot2 import \*  
  
  
class MyTest(unittest.TestCase):  
  
 def setUp(self):  
 self.love\_comp = [[True, False, False, False],  
 [False, True, False, True],  
 [False, False, False, True],  
 [False, True, False, False]]  
  
 self.work\_comp = [[True, False, True, False],  
 [True, True, True, False],  
 [False, False, True, True],  
 [True, False, False, False]]  
  
 def test\_conclusion\_true\_love(self):  
 s1 = '2'  
 s2 = '3'  
 rltn = 'Любовь'  
 self.assertEqual(config.conclusion(s1, s2, rltn), True)  
  
 def test\_conclusion\_false\_work(self):  
 s1 = '0'  
 s2 = '1'  
 rltn = 'Работа'  
 self.assertEqual(config.conclusion(s1, s2, rltn), False)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**Zodiac.feature**

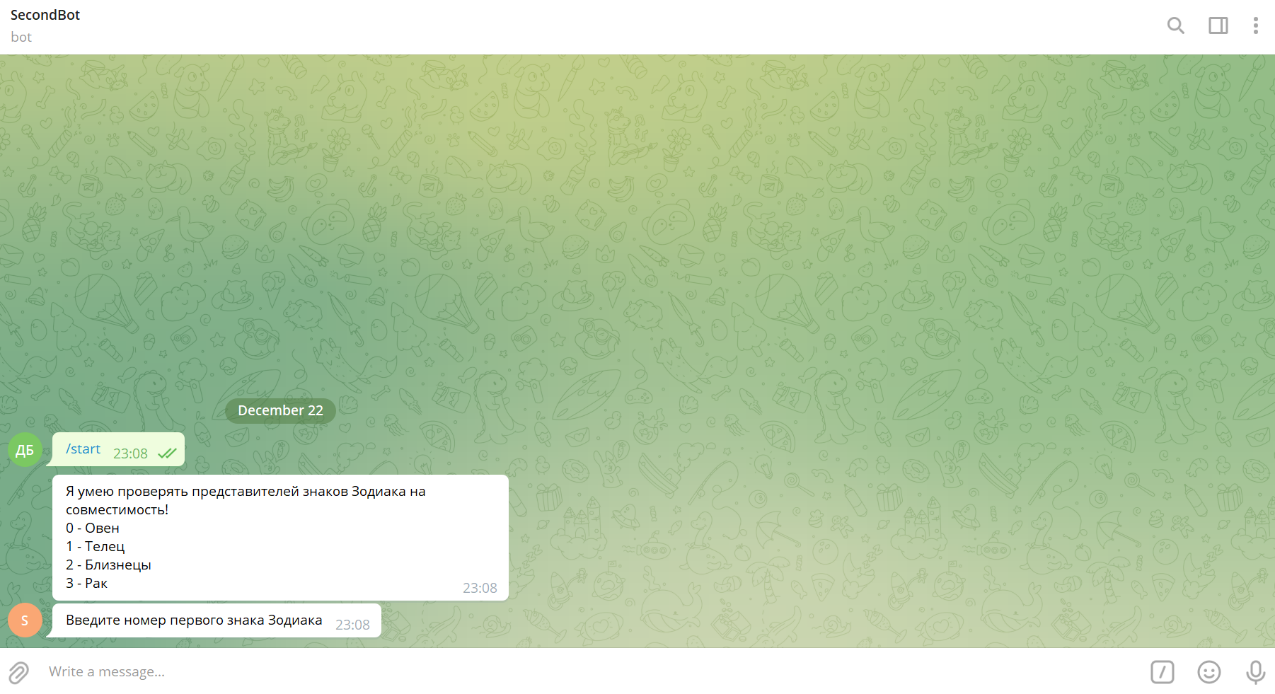
Feature: Zodiac  
 Scenario: Love compatibility  
 Given I have Zodiac signs numbers {'2'} and {'3'} and I have a relation type {'Любовь'}  
 When I check their compatibility in this relation type  
 Then I expect to get a result True  
  
 Scenario: Work compatibility  
 Given I have Zodiac signs numbers {'0'} and {'1'} and I have a relation type {'Работа'}  
 When I check their compatibility in this relation type again  
 Then I expect to get a result False

**Zodiac.py**

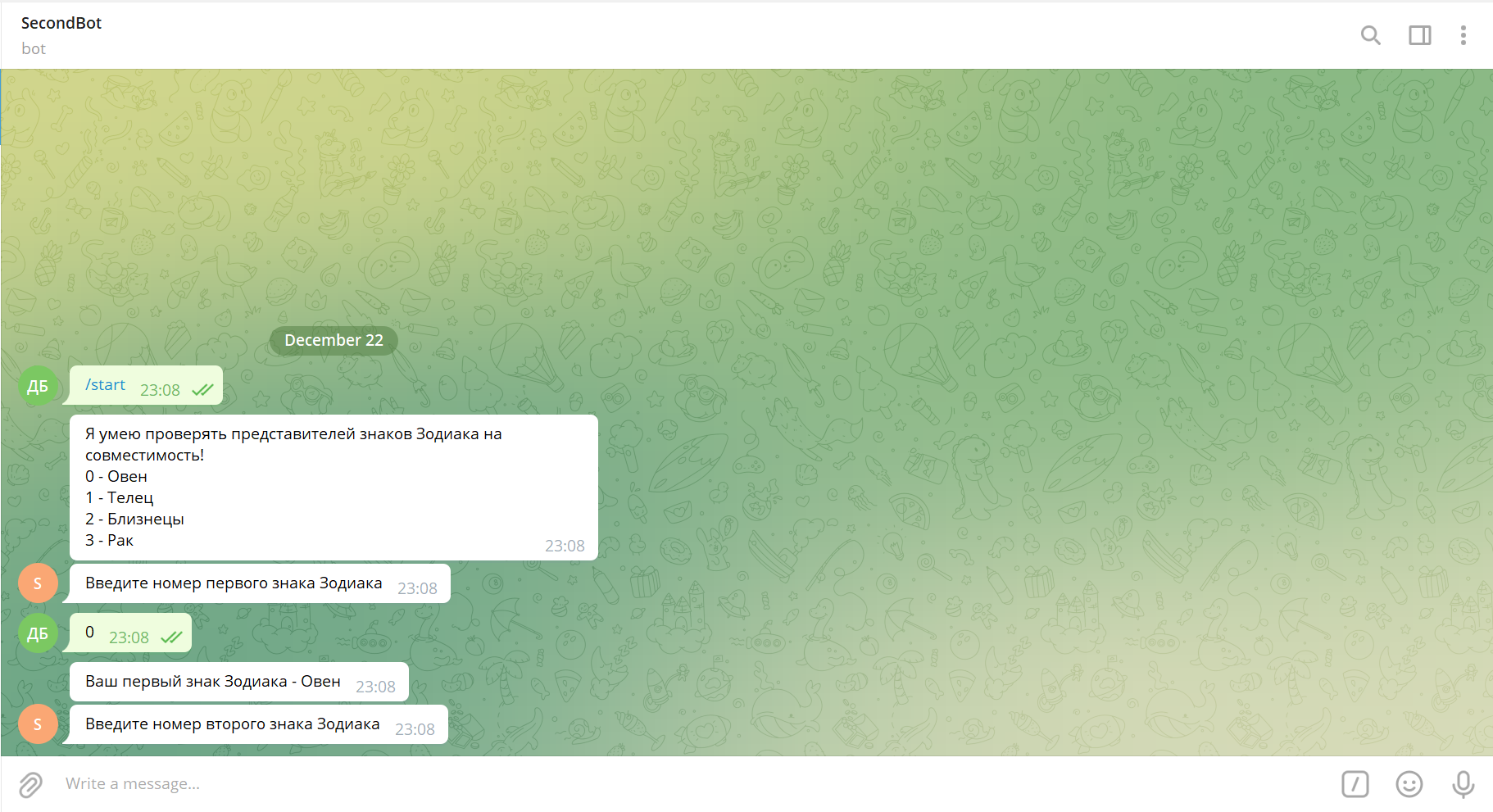
from behave import \*  
from config import \*  
  
  
@given('I have Zodiac signs numbers {\'2\'} and {\'3\'} and I have a relation type {\'Любовь\'}')  
def have\_signs(context, s1='2', s2='3', rltn='Любовь'):  
 context.s1 = s1  
 context.s2 = s2  
 context.rltn = rltn  
  
  
@when('I check their compatibility in this relation type')  
def check(context):  
 context.result = conclusion(context.s1, context.s2, context.rltn)  
  
  
@then('I expect to get a result True')  
def expect\_result(context):  
 assert context.result  
  
  
@given('I have Zodiac signs numbers {\'0\'} and {\'1\'} and I have a relation type {\'Работа\'}')  
def have\_signs(context, s1='0', s2='1', rltn='Работа'):  
 context.s1 = s1  
 context.s2 = s2  
 context.rltn = rltn  
  
  
@when('I check their compatibility in this relation type again')  
def check(context):  
 context.result = conclusion(context.s1, context.s2, context.rltn)  
  
  
@then('I expect to get a result False')  
def expect\_result(context):  
 assert not context.result

**Экранные формы с примерами выполнения программы**

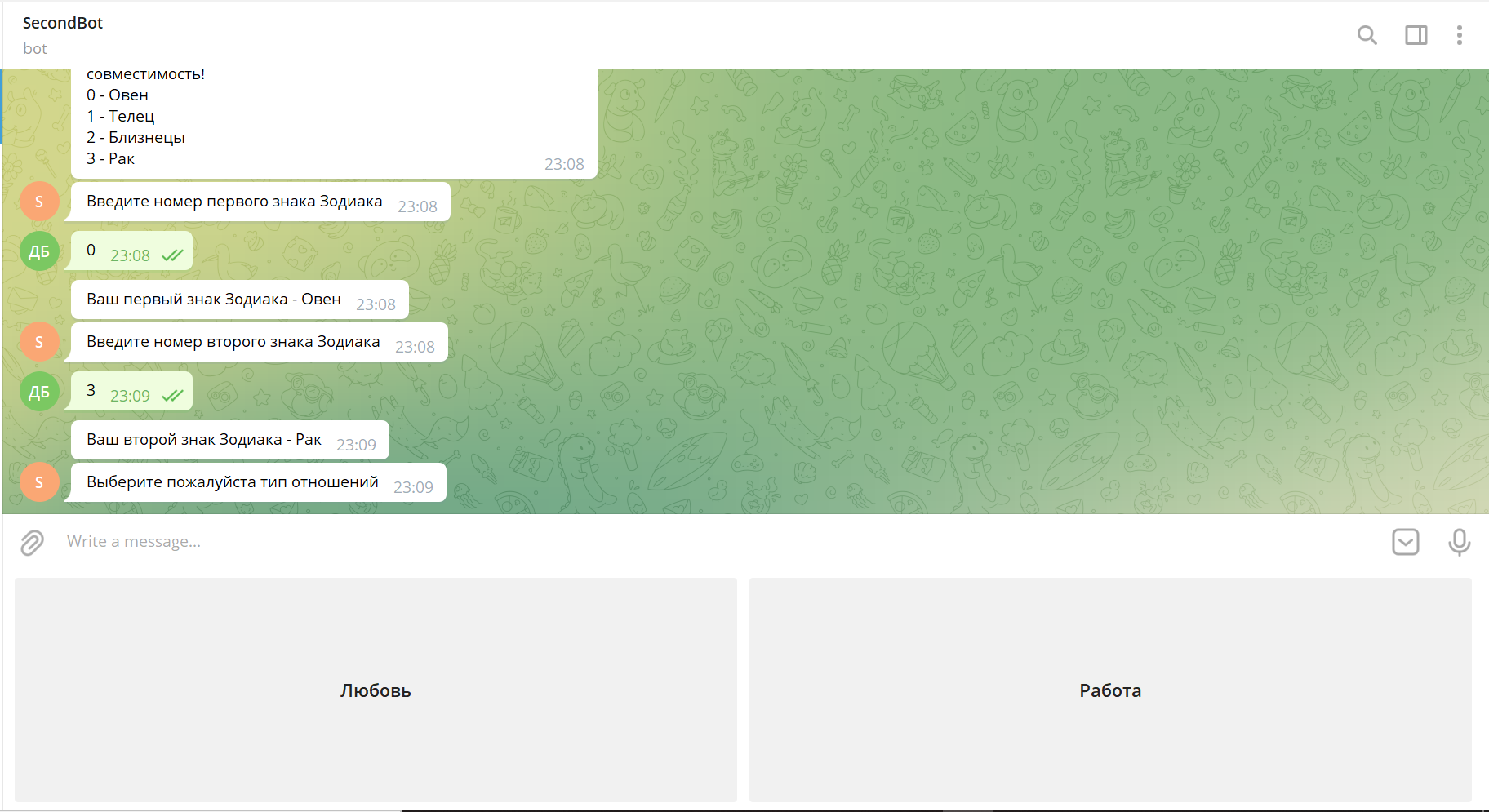
**Начало**

****

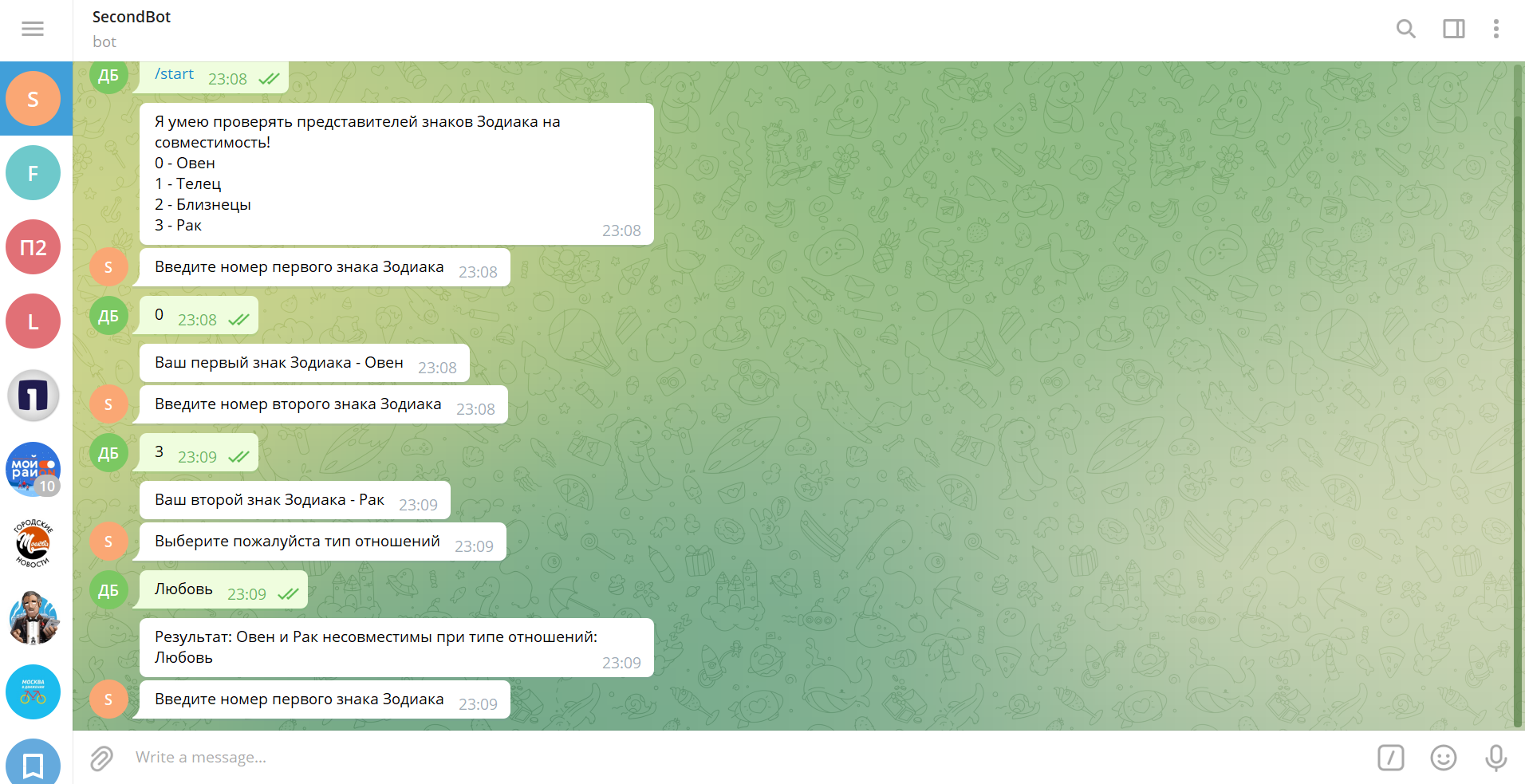
**Ввод первого номера**

****

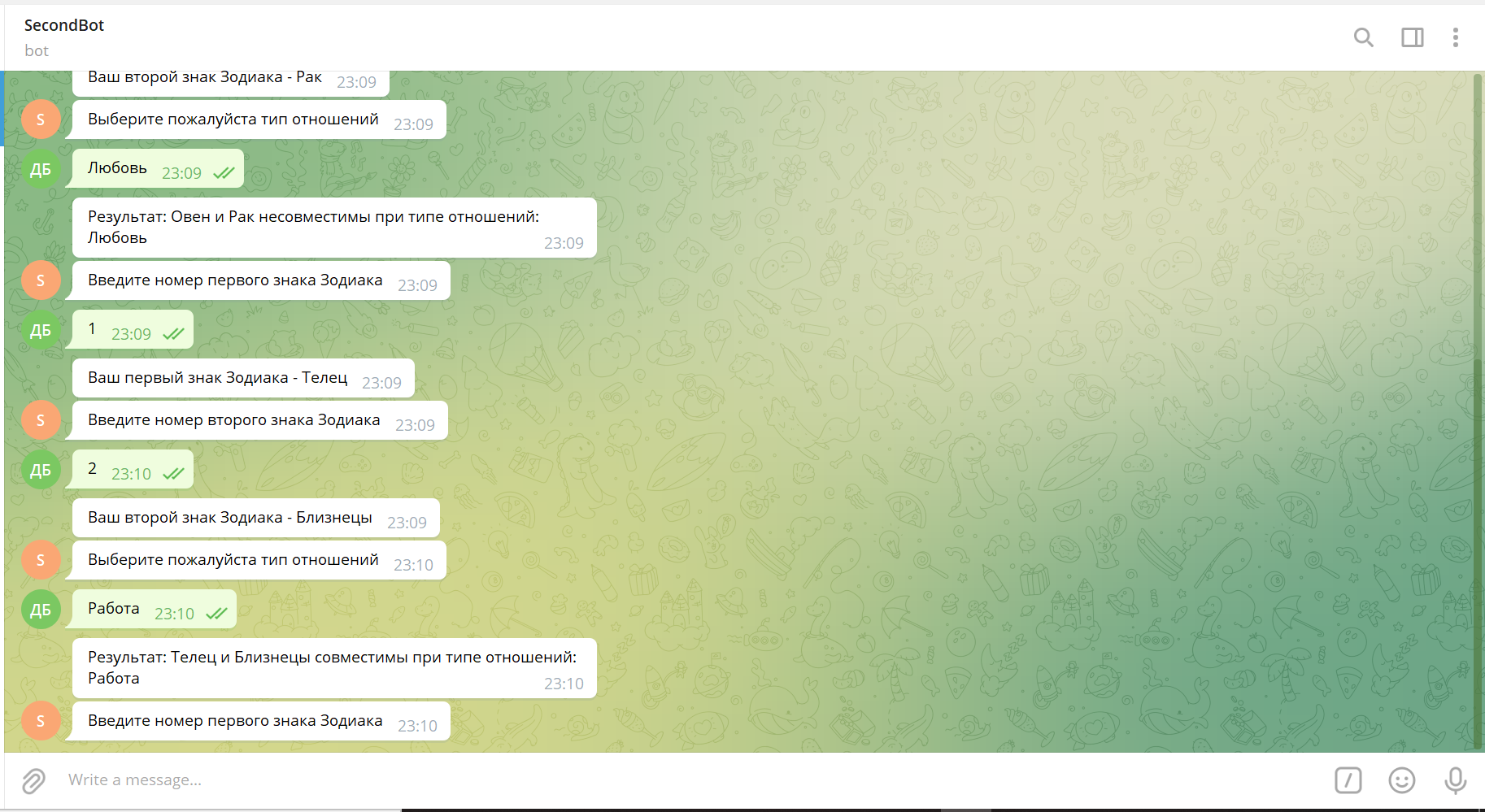
**Ввод второго номера**



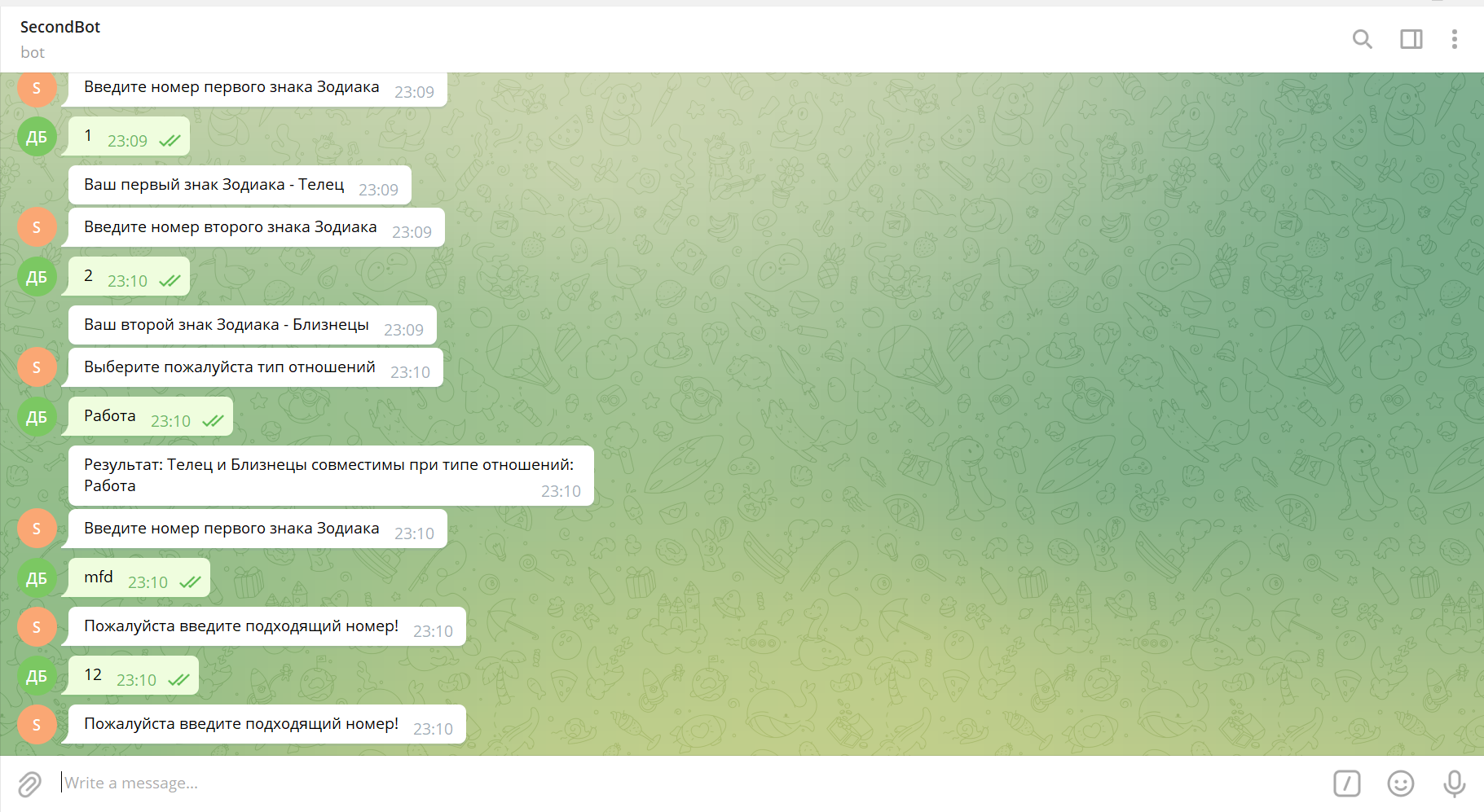
**Нажатие кнопки “Любовь”: результат - несовместимы**

****

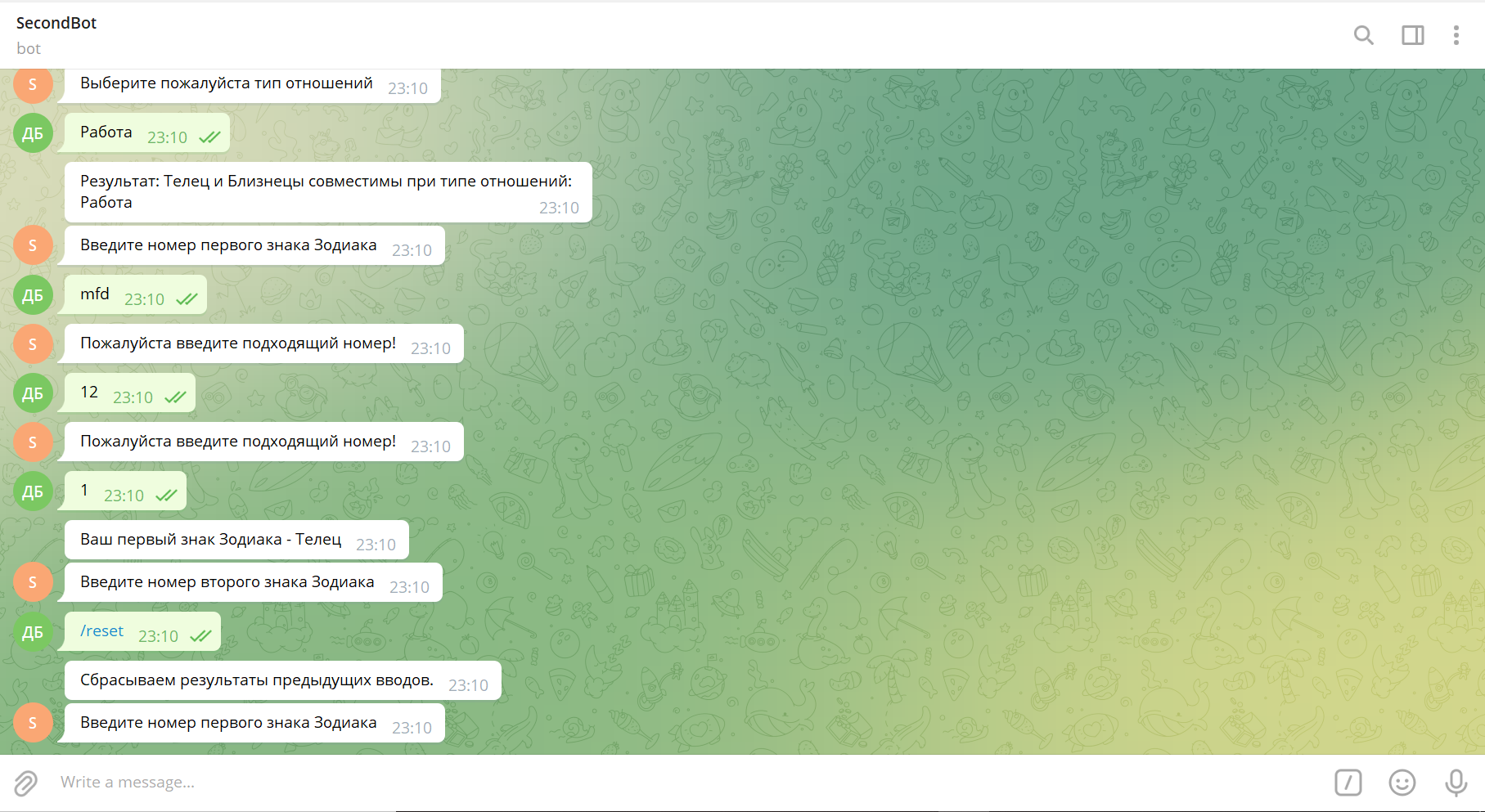
**Результат совместимости после нажатия кнопки “Работа”**

****

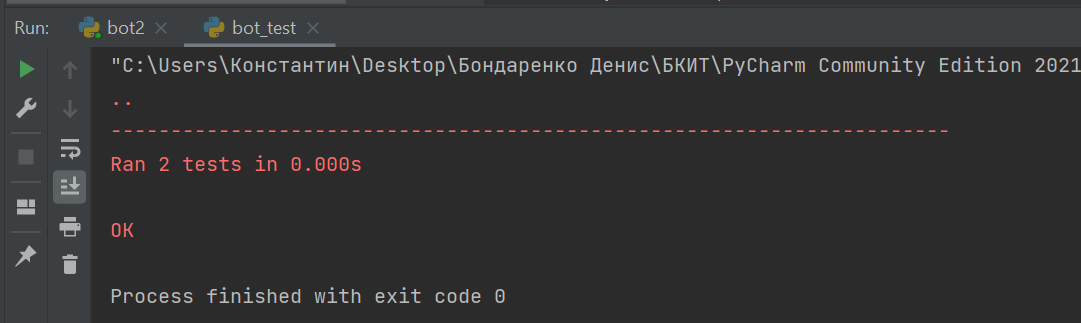
**Некорректный ввод**

****

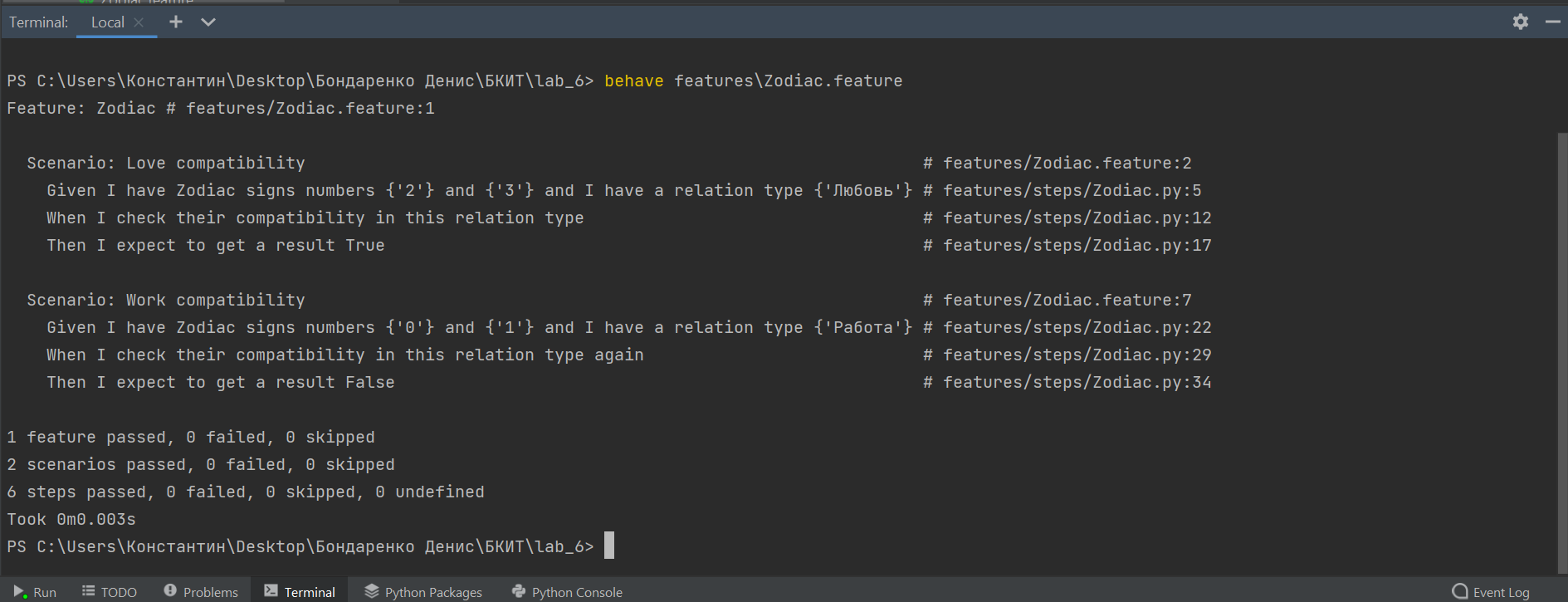
**Команда /reset**

****

**Тестирование с помощью bot\_test.py**

****

**Тестирование с помощью Zodiac.feature и Zodiac.py**

****