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:
студент группы ИУ5-31Б
Лобанов Дмитрий
Сергеевич

Подпись:_____

Дата:_____

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий
Евгеньевич

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

main.py

```
import telebot
from telebot import types

TOKEN = '***'
bot = telebot.TeleBot(TOKEN)
global config
config = ['Первое число', 'Второе число', 'Действие', 'Посчитать']
global cases
cases = ['first', 'second', 'action', 'res']
global call
call = ''
global actions
actions = ['plus', 'minus', 'multiply', 'divide', 'power']


def sum_output(config):
    config[3] = float(config[0]) + float(config[1])
    return f'{float(config[0])} + {float(config[1])} = {config[3]}'


def minus_output(config):
    config[3] = float(config[0]) - float(config[1])
    return f'{float(config[0])} - {float(config[1])} = {config[3]}'


def multiply_output(config):
    config[3] = float(config[0]) * float(config[1])
    return f'{float(config[0])} * {float(config[1])} = {config[3]}'


def power_output(config):
    config[3] = float(config[0]) ** float(config[1])
    return f'{float(config[0])} ** {float(config[1])} = {config[3]}'


def divide_output(config):
    if config[1] == '0':
        return 'Деление на ноль невозможно.'
    else:
        config[3] = float(config[0]) / float(config[1])
        return f'{float(config[0])} / {float(config[1])} = {config[3]}'


@bot.message_handler(commands='start')
def start(message):
```

```
    msg = 'Добро пожаловать.'
    markup = types.InlineKeyboardMarkup()
    btn = types.InlineKeyboardButton('Начать работу', callback_data='work')
    markup.add(btn)
    bot.send_message(message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == 'work')
def work(message):
    msg = 'Введите данные'
    markup = types.InlineKeyboardMarkup(row_width=1)
    for i in range(4):
        btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
        markup.add(btn)
    btn = types.InlineKeyboardButton('Сбросить', callback_data='reset')
    markup.add(btn)
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == cases[0] or
message.data == cases[1])
def nums(message):
    global call
    call = cases[cases.index(message.data)]
    msg = 'Введите число'
    bot.send_message(message.message.chat.id, msg)


@bot.callback_query_handler(lambda message: message.data == cases[2])
def action(message):
    msg = 'Выберите действие'
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn = types.InlineKeyboardButton('+', callback_data='plus')
    btn1 = types.InlineKeyboardButton('-', callback_data='minus')
    btn2 = types.InlineKeyboardButton('*', callback_data='multiply')
    btn3 = types.InlineKeyboardButton('/', callback_data='divide')
    btn4 = types.InlineKeyboardButton('**', callback_data='power')
    markup.add(btn, btn1, btn2, btn3, btn4)
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data in actions)
def act(message):
    if message.data == actions[0]:
        config[2] = actions[0]
    elif message.data == actions[1]:
        config[2] = actions[1]
    elif message.data == actions[2]:
        config[2] = actions[2]
    elif message.data == actions[3]:
        config[2] = actions[3]
    elif message.data == actions[4]:
        config[2] = actions[4]
    markup = types.InlineKeyboardMarkup(row_width=1)
    msg = 'Введите данные'
    for i in range(4):
        if not config[i].isdigit() and not config[i] in actions:
                btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
                markup.add(btn)
    btn = types.InlineKeyboardButton('Сбросить', callback_data='reset')
    markup.add(btn)
```

```
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == cases[3])
def res(message):
    if config[0].isdigit() and config[1].isdigit() and (config[2] in actions):
        if config[2] == 'plus':
            msg = sum_output(config)
        elif config[2] == 'minus':
            msg = minus_output(config)
        elif config[2] == 'multiply':
            msg = multiply_output(config)
        elif config[2] == 'divide':
            msg = divide_output(config)
        elif config[2] == 'power':
            msg = power_output(config)
        markup = types.InlineKeyboardMarkup()
        btn = types.InlineKeyboardButton('Сбросить', callback_data='reset')
        markup.add(btn)
        bot.send_message(message.message.chat.id, msg, reply_markup=markup)
    else:
        msg = 'Недостаточно данных'
        markup = types.InlineKeyboardMarkup()
        for i in range(4):
            if not config[i].isdigit() and not config[i] in actions:
                btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
                markup.add(btn)
        btn = types.InlineKeyboardButton('Сбросить', callback_data='reset')
        markup.add(btn)
        bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == 'reset')
def reset(message):
    msg = 'Данные сброшены.'
    global config
    config = ['Первое число', 'Второе число', 'Действие', 'Посчитать']
    markup = types.InlineKeyboardMarkup()
    btn = types.InlineKeyboardButton('Продолжить', callback_data='work')
    markup.add(btn)
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.message_handler(content_types='text')
def text(message):
    if (call == cases[0] or call == cases[1]) and message.text.isdigit():
        if call == cases[0]:
            config[0] = message.text
            bot.send_message(message.chat.id, f'Вы ввели первое число
{float(message.text)}')
        elif call == cases[1]:
            config[1] = message.text
            bot.send_message(message.chat.id, f'Вы ввели второе число
{float(message.text)}')
        markup = types.InlineKeyboardMarkup(row_width=1)
        msg = 'Введите данные'
        for i in range(4):
            if not config[i].isdigit() and not config[i] in actions:
                btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
                markup.add(btn)
        btn = types.InlineKeyboardButton('Сбросить', callback_data='reset')
```

```
        markup.add(btn)
        bot.send_message(message.chat.id, msg, reply_markup=markup)
    if (call == cases[0] or call == cases[1]) and not message.text.isdigit():
        msg = 'Ошибка, введите число'
        bot.send_message(message.chat.id, msg)


# bot.polling(none_stop=True)
```

tests.py

```
from main import sum_output, minus_output, divide_output
import unittest


class Tests(unittest.TestCase):

    def test_sum(self):
        msg = sum_output(['1', '2', 'plus', 'Посчитать'])
        self.assertEqual('1.0 + 2.0 = 3.0', msg)

    def test_minus(self):
        msg = minus_output(['3', '2', 'minus', 'Посчитать'])
        self.assertEqual('3.0 - 2.0 = 1.0', msg)

    def test_divide(self):
        msg = divide_output(['4', '2', 'divide', 'Посчитать'])
        self.assertEqual('4.0 / 2.0 = 2.0', msg)
```

bddM.feature

```
Feature: multuply
  Scenario: multiply 3 and 4
    Given I have context for multiplying: ['3', '4', 'multiply', 'Посчитать']
    When I call multiply_output
    Then I expect to get message with multiplying result: '3.0 * 4.0 = 12.0'
```

bddP.feature

```
Feature: power
  Scenario: 3 power 4
    Given I have context for powering: ['3', '4', 'power', 'Посчитать']
    When I call power_output
    Then I expect to get message with power result: '3.0 ** 4.0 = 81.0'
```

stepsM.py

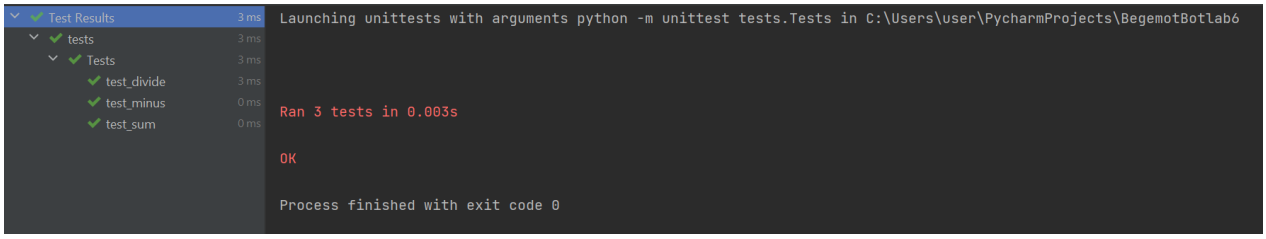
```
# -*- coding: utf-8 -*-
import string
from main import *
from behave import given, when, then


@given(u'I have context for multiplying: [\'{first}\', \'{second}\',
\'{action}\', \'{result}\']')
def step_mult(context, first: string, second: string, action: string, result:
string):
    context.first = first
    context.second = second
    context.action = action
    context.result = result


@when(u'I call multiply_output')
def step_mult(context):
    context.msg = multiply_output([context.first, context.second,
context.action, context.result])


@then(u'I expect to get message with multiplying result: \'{msg}\'')
def step_mult(context, msg: string):
    assert context.msg == msg
```

steps.py

```
# -*- coding: utf-8 -*-
import string
from main import *
from behave import given, when, then


@given(u'I have context for powering: [\'{first}\', \'{second}\',
\'{action}\', \'{result}\']')
def step_pow(context, first: string, second: string, action: string, result:
string):
    context.first = first
    context.second = second
    context.action = action
    context.result = result


@when(u'I call power_output')
def step_pow(context):
    context.msg = power_output([context.first, context.second, context.action,
context.result])


@then(u'I expect to get message with power result: \'{msg}\'')
def step_pow(context, msg: string):
    assert context.msg == msg
```

Примеры выполнения программы

tests.py



behave