**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 |
| Кондрахин Сергей Сергеевич |  | Гапанюк Юрий  Евгеньевич |
|  |  |  |

Москва, 2021 г.

**Задание**

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', 'Третье число', 'Операции', 'Посчитать выражение']  
global cases  
cases = ['first', 'second', 'third', 'action', 'result']  
global call  
call = ''  
global actions  
actions = ['plus', 'multiplication']  
  
def plus\_out\_put(config):  
 config[4] = float(config[0]) + float(config[1]) + float(config[2])  
 return f'{float(config[0])} + {float(config[1])} + {float(config[2])} = {config[4]}'  
  
def multiplication\_out\_put(config):  
 config[4] = float(config[0]) \* float(config[1]) \* float(config[2])  
 return f'{float(config[0])} \* {float(config[1])} \* {float(config[2])} = {config[4]}'  
  
@bot.message\_handler(commands='start')  
def start(message):  
 msg = 'Добро пожаловать'  
 markup = types.InlineKeyboardMarkup()  
 btn = types.InlineKeyboardButton('Начать работу с калькулятором', 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(5):  
 btn = types.InlineKeyboardButton(f'{config[i]}', callback\_data=f'{cases[i]}')  
 markup.add(btn)  
 btn = types.InlineKeyboardButton('Сбросить данные', 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] or message.data == cases[2])  
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[3])  
def action(message):  
 msg = 'Выберите операцию'  
 markup = types.InlineKeyboardMarkup(row\_width=2)  
 btn = types.InlineKeyboardButton('+', callback\_data='plus')  
 btn1 = types.InlineKeyboardButton('\*', callback\_data='multiplication')  
 markup.add(btn, btn1)  
 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[3] = actions[0]  
 elif message.data == actions[1]:  
 config[3] = actions[1]  
 elif message.data == actions[2]:  
 config[3] = actions[2]  
 else:  
 config[3] = actions[3]  
 markup = types.InlineKeyboardMarkup(row\_width=1)  
 msg = 'Введите данные'  
 for i in range(5):  
 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 данные', 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[4])  
def result(message):  
 if config[0].isdigit() and config[1].isdigit() and config[2].isdigit() and (config[3] in actions):  
 if config[3] == 'plus':  
 msg = plus\_out\_put(config)  
 else:  
 msg = multiplication\_out\_put(config)  
 markup = types.InlineKeyboardMarkup()  
 btn = types.InlineKeyboardButton('Сбросить данные', 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(5):  
 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 данные', 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', 'Третье число', 'Операции', 'Посчитать выражение']  
 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] or call == cases[2]) 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)}')  
 elif call == cases[2]:  
 config[2] = message.text  
 bot.send\_message(message.chat.id, f'Вы ввели третье число {float(message.text)}')  
 markup = types.InlineKeyboardMarkup(row\_width=1)  
 msg = 'Введите данные'  
 for i in range(5):  
 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 данные', callback\_data='reset')  
 markup.add(btn)  
 bot.send\_message(message.chat.id, msg, reply\_markup=markup)  
 if (call == cases[0] or call == cases[1] or call == cases[2]) and not message.text.isdigit():  
 msg = 'Ошибка, попробуйте еще раз'  
 bot.send\_message(message.chat.id, msg)  
  
# bot.polling(none\_stop=True)

**tests.py**

from main import plus\_out\_put, multiplication\_out\_put  
import unittest  
  
class Tests(unittest.TestCase):  
 def test\_plus(self):  
 msg = plus\_out\_put(['1.0', '2.0', '3.0', 'plus', 'Посчитать выражение'])  
 self.assertEqual('1.0 + 2.0 + 3.0 = 6.0', msg)  
  
 def test\_multiplication(self):  
 msg = multiplication\_out\_put(['3.0', '4.0', '5.0', 'multiplication', 'Посчитать выражение'])  
 self.assertEqual('3.0 \* 4.0 \* 5.0 = 60.0', msg)

**stepsPlus.py**

# -\*- coding: utf-8 -\*-  
import string  
from main import \*  
from behave import given, when, then  
  
@given(u'I have context for plusing: [\'{first}\', \'{second}\', \'{third}\', \'{action}\', \'{result}\']')  
def step\_plus(context, first: string, second: string, third: string, action: string, result: string):  
 context.first = first  
 context.second = second  
 context.third = third  
 context.action = action  
 context.result = result  
  
@when(u'I call plus\_out\_put')  
def step\_plus(context):  
 context.msg = plus\_out\_put([context.first, context.second, context.third, context.action, context.result])  
  
@then(u'I expect to get message with plusing result: \'{msg}\'')  
def step\_plus(context, msg: string):  
 assert context.msg == msg

**stepsMultiplication.py**

# -\*- coding: utf-8 -\*-  
import string  
from main import \*  
from behave import given, when, then  
  
@given(u'I have context for multiplicationing: [\'{first}\', \'{second}\', \'{third}\', \'{action}\', \'{result}\']')  
def step\_multiplication(context, first: string, second: string, third: string, action: string, result: string):  
 context.first = first  
 context.second = second  
 context.third = third  
 context.action = action  
 context.result = result  
  
@when(u'I call multiplication\_out\_put')  
def step\_multiplication(context):  
 context.msg = multiplication\_out\_put([context.first, context.second, context.third, context.action, context.result])  
  
@then(u'I expect to get message with multiplicationing result: \'{msg}\'')  
def step\_multiplication(context, msg: string):  
 assert context.msg == msg

**bddPlus.feature**

Feature: plus  
 Scenario: plus 5 and 11 and 15  
 Given I have context for plusing: ['**5.0**', '**11.0**', '**15.0**', '**plus**', '**Посчитать выражение**']  
 When I call plus\_out\_put  
 Then I expect to get message with plusing result: '**5.0 + 11.0 + 15.0 = 31.0**'

**bddMultiplication.feature**

Feature: multiplication  
 Scenario: multiplication 5 and 11 and 15  
 Given I have context for multiplicationing: ['**5.0**', '**11.0**', '**15.0**', '**multiplication**', '**Посчитать выражение**']  
 When I call multiplication\_out\_put  
 Then I expect to get message with multiplicationing result: '**5.0 \* 11.0 \* 15.0 = 825.0**'

**Примеры выполнения программы**



