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 «ГУИМЦ»

Кафедра ИУ5 «Системы обработки информации и управления»

Дисциплина «Базовые компоненты ИТ»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1-6 И ДЗ

Студент: Печуркин Д.С., группа ИУ5Ц-51Б

Преподаватель: Гапанюк Ю.Е.

2022г.

Лабораторная работа № 1 и 5

Задание:

Разработать программу для решения биквадратного уравнения.

Программа должна быть разработана в виде консольного приложения на языке Python.

Программа осуществляет ввод с клавиатуры коэффициентов A, B, C, вычисляет дискриминант и ДЕЙСТВИТЕЛЬНЫЕ корни уравнения (в зависимости от дискриминанта).

Коэффициенты A, B, C могут быть заданы в виде параметров командной строки (вариант задания параметров приведен в конце файла с примером кода). Если они не заданы, то вводятся с клавиатуры в соответствии с пунктом 2. Описание работы с параметрами командной строки.

Если коэффициент A, B, C введен или задан в командной строке некорректно, то необходимо проигнорировать некорректное значение и вводить коэффициент повторно пока коэффициент не будет введен корректно. Корректно заданный коэффициент - это коэффициент, значение которого может быть без ошибок преобразовано в действительное число.

Задание:

1. Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4.
2. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.
3. Разработайте модульные тесты. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк (не менее 3 тестов).
    - BDD - фреймворк (не менее 3 тестов).
    - Создание Mock-объектов (необязательное дополнительное задание).

Программа:

```
import sys
import unittest
from unittest.mock import Mock, patch


def get_coef(index, prompt):
    '''
    Читаем коэффициент из командной строки или вводим с клавиатуры
    Args:
        index (int): Номер параметра в командной строке
        prompt (str): Приглашение для ввода коэффицента
    Returns:
        float: Коэффициент квадратного уравнения
    '''

    try:
        # Пробуем прочитать коэффициент из командной строки
        coef_str = sys.argv[index]
        coef = float(coef_str)
    except:
        # Проверяем на число
        while True:
            try:
                # Вводим с клавиатуры
                print(prompt, end="")
                coef_str = input()
                # Переводим строку в действительное число
                coef = float(coef_str)
            except:
```

```
            print("Введите число!")
        else:
            break

    return coef

def get_roots(a, b, c):
    '''
    Вычисление корней квадратного уравнения
    Args:
        a (float): коэффициент A
        b (float): коэффициент B
        c (float): коэффициент C
    Returns:
        result[float]: Список корней
    '''

    preresult = []
    result = set()
    D = b*b - 4*a*c

    if D == 0:
        preresult.append(-b/2*a)

    elif D > 0:
        root1 = (-b - D**0.5) / (2 * a)
        root2 = (-b + D**0.5) / (2 * a)

        if root1 >= 0:
            preresult.append(root1)
```

```
        if root2 >= 0:
            preresult.append(root2)

    for root in preresult:
        result.add(root ** 0.5)
        result.add(-root ** 0.5)

    return result


def main():
    """
    Основная фунция
    """

    a = get_coef(1, "Введите a: ")

    if not a:
        print("Это не биквадратное уравнение!")
        return

    b = get_coef(2, "Введите b: ")
    c = get_coef(3, "Введите c: ")

    roots = list(get_roots(a, b, c))

    len_roots = len(roots)

    if len_roots == 0:
        print("Корней нет")
```

```
        if len_roots == 1:
            print(f"Единственный корень: {roots[0]}")

        if len_roots == 2:
            print(f"Есть два корня: {roots[0]} и {roots[1]}")

        if len_roots == 3:
            print(f"Есть три корня: {roots[0]}, {roots[1]} и {roots[2]}")

        if len_roots == 4:
            print(f"Есть четыре корня: {roots[0]}, {roots[1]}, {roots[2]} и {roots[3]}")

    class get_roots_TDD(unittest.TestCase):
        def test(self):
            self.assertEqual(get_roots(1, -4, 0), set())


            self.assertEqual(get_roots(1, -5, 6), {1.4142135623730951,
                                                   -1.7320508075688772,
                                                   1.7320508075688772,
                                                   -1.4142135623730951})

            self.assertEqual(get_roots(-4, 16, 0), {0, 2, -2})
            self.assertRaises(TypeError, get_roots, 1 + 1j, 2 - 3j, 5 + 10j)

            # C Mock - объектами

            roots_obj = Roots4()
```

```
        roots_obj.roots = Mock(return_value=[5, 1, 8])

        print(roots_obj.roots(-4, 16, 0))

        roots_obj.roots.assert_called_once()
        roots_obj.roots.assert_called_with(-4, 16, 0)

        self.assertEqual(roots_obj.roots(1, 2, 3), set())

class Roots4:
    def __init__(self):
        pass

    def roots(self, a, b, c):
        return get_roots(a, b, c)

if __name__ == '__main__':
    unittest.main()
    # main()
```

Лабораторная работа №2.

Задание:

Необходимо создать виртуальное окружение и установить в него хотя бы один внешний пакет с использованием pip.

Необходимо разработать программу, реализующую работу с классами. Программа должна быть разработана в виде консольного приложения на языке Python 3.

Все файлы проекта (кроме основного файла main.py) должны располагаться в пакете lab_python_oop.

Каждый из нижеперечисленных классов должен располагаться в отдельном файле пакета lab_python_oop.

Абстрактный класс «Геометрическая фигура» содержит абстрактный метод для вычисления площади фигуры. Подробнее про абстрактные классы и методы Вы можете прочитать здесь.

Класс «Цвет фигуры» содержит свойство для описания цвета геометрической фигуры. Подробнее про описание свойств Вы можете прочитать здесь.

Класс «Прямоугольник» наследуется от класса «Геометрическая фигура». Класс должен содержать конструктор по параметрам «ширина», «высота» и «цвет». В конструкторе создается объект класса «Цвет фигуры» для хранения цвета. Класс должен переопределять метод, вычисляющий площадь фигуры.

Класс «Круг» создается аналогично классу «Прямоугольник», задается параметр «радиус». Для вычисления площади используется константа math.pi из модуля math.

Класс «Квадрат» наследуется от класса «Прямоугольник». Класс должен содержать конструктор по длине стороны. Для классов «Прямоугольник», «Квадрат», «Круг»:

Определите метод "repr", который возвращает в виде строки основные параметры фигуры, ее цвет и площадь. Используйте метод format - https://pyformat.info/

Название фигуры («Прямоугольник», «Квадрат», «Круг») должно задаваться в виде поля данных класса и возвращаться методом класса.

В корневом каталоге проекта создайте файл main.py для тестирования Ваших классов (используйте следующую конструкцию - https://docs.python.org/3/library/__main__.html). Создайте следующие объекты и выведите о них информацию в консоль (N - номер Вашего варианта по списку группы):

Прямоугольник синего цвета шириной N и высотой N.

Круг зеленого цвета радиусом N.

Квадрат красного цвета со стороной N.

Также вызовите один из методов внешнего пакета, установленного с использованием pip.

Программа:

```
from lab_python_oop.box import *
from lab_python_oop.rectangle import *
from lab_python_oop.circle import *
import numpy as np

def main():
    r = Rectangle("синего", 2, 2)
    c = Circle("зеленого", 2)
    s = Box("красного", 2)
    print(r)
    print(c)
    print(s)

    mas = np.array([[1,5,6,5],
                    [3,9,-2,0],
                    [9,4,7,1],
```

```
                [1,0,9,4]])

    print(f"\n{np.linalg.det(mas)}")
    print(mas.dot(mas))

if __name__ == '__main__':
    main()
```

Классы соответственно объявлены в других файлах.

Лабораторная работа 3-4.

Описание задания

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp. Решение каждой задачи должно располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задача 1 (файл field.py)

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

В качестве первого аргумента генератор принимает список словарей, дальше через *args генератор принимает неограниченное количество аргументов.

Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.

Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for i in items:
            yield i.get(args[0])
    else:
        for i in items:
            d = {}
```

```
                for a in args:
                    if i.get(a) is not None:
                        d[a] = i.get(a)
                yield d


goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
]
```

Задача 2 (файл gen_random.py)

Необходимо реализовать генератор gen_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.

```
9 lines (4 sloc)  130 Bytes

import random


def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randint(begin, end)
```

Задача 3 (файл unique.py)

Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.

Конструктор итератора также принимает на вход именованный bool-параметр ignore_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.

При реализации необходимо использовать конструкцию **kwargs.

Итератор должен поддерживать работу как со списками, так и с генераторами.

Итератор не должен модифицировать возвращаемые значения.

```
class Unique(object):
    def __init__(self, items, ignore_case=False, **kwargs):

        self.items = items
        self.case = ignore_case
        self.kwargs = kwargs
        self.MySet = set()

        pass

    def __next__(self):
```

```
        iterator = iter(self.items)
        while(True):
            try:
                obj = next(iterator)
            except StopIteration:
                raise
            # else:
            #     if obj not in self.MySet and self.case==False:
            #         self.MySet.add(obj)
            #         return obj
            #     elif self.case==True:
            #         a = str(obj)
            #         if a.lower() not in self.MySet:
            #             self.MySet.add(a)
            #             return a

            else:
                if self.case == True and isinstance(obj, str):
                    a = str(obj)
                    if a.lower() not in self.MySet:
                        self.MySet.add(a.lower())
                        return obj
                elif obj not in self.MySet:
                    self.MySet.add(obj)
                    return obj
    def __iter__(self):
        return self
```

Задача 5 (файл print_result.py)

Необходимо реализовать декоратор print_result, который выводит на экран результат выполнения функции.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.

Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

```
def print_result(func):
    def dec_func(*args):
        print(func.__name__)
        result = func(*args)
        print(result)
        return result
    return dec_func


@print_result
def test_1():
    return 1
```

```
@print_result
def test_2():
    return 'iu5'


@print_result
def test_3():
    return {'a': 1, 'b': 2}


@print_result
def test_4():
    return [1, 2]
#
```

Задача 6 (файл cm_timer.py)

Необходимо написать контекстные менеджеры cm_timer_1 и cm_timer_2, которые считают время работы блока кода и выводят его на экран.

После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

cm_timer_1 и cm_timer_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

```
import time
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        self.start_t = None

    def __enter__(self):
        self.start_t = time.time()

    def __exit__(self,x,y,z):
        print(f'time: {time.time() - self.start_t}')


@contextmanager
def cm_timer_2():
    pass
```

Задача 7 (файл process_data.py)

В файле data_light.json содержится фрагмент списка вакансий.

Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет

декоратора @print_result печатается результат, а контекстный менеджер cm_timer_1 выводит время работы цепочки функций.

Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.

Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.

Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.

Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python. Для модификации используйте функцию map.

Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

`

. . .

```
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.print_result import test_1,test_2,test_3,test_4
from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.fieldd import field
from lab_python_fp.process_data import f1,f2,f3,f4
from contextlib import contextmanager
import json
import time



def main():
    print("1.Field:")
    goods = [
          {'title': 'Ковер', 'price': 2000, 'color': 'green'},
          {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]


    print(*field(goods, 'title'))
```

```
print(*field(goods, 'title', 'price'))


print("2.Gen_Random:", *gen_random(10, 2, 100))

print("3.Unique:")
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data1 = ["a", "A", "b", "B", "a", "A", "b", "B"]
data3 = gen_random(10, 1, 4)
print(*Unique(data))
print(*Unique(data1))
print(*Unique(data1, True))
print(*Unique(data3))

print("4.Sort:")
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
result = sorted(data, key=abs, reverse=True)

result_with_lambda = sorted(data,key = lambda a: a if a > 0 else -a, reverse=True)

print(result)
print(result_with_lambda)

print("5.Print_result:")
print('!!!!!!!!')
test_1()
test_2()
test_3()
test_4()

#print("6.Cm_timer:")
#with cm_timer_1():
 #   time.sleep(5.5)

#print("7.process_data:")
path = 'data_light2.json'

path2 = 'data_light.json'
with open(path2, encoding="utf-8") as f:
    data_j = json.load(f)

#F1=f1(data_j)
```

```
    #f3(f2(f1(data_j)))
    with cm_timer_1():
        f4(f3(f2(f1(data_j))))



if __name__ == '__main__':
    main()
```

Результат:

```
1.Field:
Ковер Диван для отдыха
{'title': 'Ковер', 'price': 2000} {'title': 'Диван для отдыха', 'price': 5300}
2.Gen_Random: 38 52 33 65 58 57 51 85 58 33
3.Unique:
1 2
a A b B
a b
2 3 4
4.Sort:
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
5.Print_result:
!!!!!!!!
test_1
1
test_2
iu5
test_3
{'a': 1, 'b': 2}
test_4
[1, 2]
f1
['1C программист', '2-ой механик', '3-ий механик', '4-ый механик', '4-ый электромеханик', '[химик-эксперт', 'ASIC специалист', 'JavaScript разработчик', 'RTL специалист', 'Web-программист', 'web-разработчик', 'Автожестянщик', 'Автоинструктор', 'Автомаляр', 'Автомойщик', 'Автор студенческих работ по различным дисциплинам', 'автослесарь', 'Автослесарь - моторист', 'Автоэлектрик', 'Агент', 'Агент банка', 'Агент нпф', 'Агент по гос. закупкам недвижимости', 'Агент по недвижимости', 'Агент по недвижимости (стажер)',
```

'Агент по недвижимости / Риэлтор', 'Агент по привлечению юридических лиц', 'Агент по продажам (интернет, ТВ, телефония) в ПАО Ростелеком в населенных пунктах Амурской области: г. Благовещенск, г. Белогорск, г. Свободный, г. Шимановск, г. Зея, г. Тында', 'Агент торговый', 'агрегатчик-топливник KOMATSU', 'агроном', 'агроном по защите растений', 'Агроном-полевод', 'агрохимик почвовед', 'Администратор', 'Администратор (удаленно)', 'Администратор Active Directory', 'Администратор в парикмахерский салон', 'Администратор зала (предприятий общественного питания)', 'Администратор кофейни', 'Администратор на ресепшен', 'Администратор на телефоне', 'Администратор по информационной безопасности', 'Администратор ресторана', 'Администратор сайта', 'Администратор ярмарок выходного дня', 'Администратор-кассир', 'Аккомпаниатор на 0,5 ст.', 'аккумуляторщик 4 разряда', 'Акушерка', 'акушерка в родильное отделение', 'Акушерка женской консультации', 'Акушерка Лысогорская врачебная амбулатория', 'Акушерка ФАП', 'Акушерка, АО', 'Акушерка, ВП', 'Альпинист промышленный', 'Аналитик', 'Анестезиолог - реаниматолог', 'анестезиолог-реаниматолог', 'анестезиолог-реаниматолог детский', 'аниматор', 'антенщик-мачтовик 4 разряда', 'аппаратчик обработки зерна', 'Аппаратчик обработки зерна 5 разряда', 'Аппаратчик пастеризации', 'Аппаратчик установки опытного производства', 'Аппаратчик химводоочистки', 'Арматурщик', 'арматурщик кузовного цеха', 'арматурщики', 'Артист (кукловод) театра кукол', 'Артист оркестра', 'Артист отдела социально - культурной деятельности Районного ЦНК', 'Артист хора', 'артист(кукловод) театра кукол', 'Артист-вокалист (солист)', 'артист-кукловод', 'архивариус', 'Архивариус (Орехово-Зуевский филиал)', 'Архитектор, картограф, инженер-проектировщик', 'Ассистент главы отделения', 'Ассистент отдела продаж', 'Ассистент режиссера', 'балетмейстер-постановщик', 'Бармен', 'Бармен-кассир в кафе', 'Бармен-официант', 'Бетонщик', 'Бетонщик (на срубку свай)', 'Бетонщик - арматурщик', 'Бетонщик-монолитчик', 'Библиограф', 'Библиотекарь', 'Библиотекарь отдела абонемента', 'Биолог', 'Боец скота', 'Бригадир в животноводстве', 'бригадир животноводства', 'бригадир мобильной бригады', 'Бригадир технического обслуживания газоиспользующего оборудования', 'Бригадир, производитель работ', 'Брокер коммерческой недвижимости', 'Брошюровщик', 'Бухгалтер', 'Бухгалтер (по заработной плате)', 'Бухгалтер 2 категории', 'Бухгалтер на группу "Обработка первичной документации"', 'Бухгалтер по ведению первичной документации', 'Бухгалтер по заработной плате', 'Бухгалтер по МТП и ГСМ', 'Бухгалтер по начислению заработной платы', 'Бухгалтер по расчету заработной платы', 'Бухгалтер по расчету калькуляции', 'Бухгалтер, ведущий', 'Бухгалтер,Ведущий бухгалтер', 'БУХГАЛТЕР-Делопроизводитель', 'бухгалтер-кассир', 'Бухгалтер-кассир 1 категории', 'Бухгалтер-материалист', 'Бухгалтер-ревизор', 'Бухгалтер-экономист', 'Вальцовщик', 'варщик зефира', 'варщик мармеладных изделий', 'Вахта', 'Вахтер', 'Вахтёр', 'Веб - программист

(PHP, JS) / Web разработчик', 'веб-дизайнер', 'Веб-программист', 'ведущий агрохимик лаборатории полевых изысканий отдела агроэкологического мониторинга почв', 'Ведущий библиотекарь отдела книгохранения',

Домашнее задание

Цель домашнего задания: изучение возможностей создания ботов в Telegram и их тестирования.

Задание:

Модифицируйте код лабораторной работы №5 или №6 таким образом, чтобы он был пригоден для модульного тестирования.

Используя материалы лабораторной работы №4 создайте модульные тесты с применением TDD - фреймворка (4 теста).

```
import unittest
import telebot
import requests
from telebot import types

bot = telebot.TeleBot('TELEGRAM-API')
appid = 'OPEN-WEATHER-API'
s_city = "Moscow (RU)"
city_id = 0



@bot.message_handler(commands=['start'])
def start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    btn1 = types.KeyboardButton("Погода")
    btn2 = types.KeyboardButton("Температура")
    markup.add(btn1, btn2)
    if type(message) != str:
        bot.send_message(message.chat.id, text="Я родился!",
reply_markup=markup)
    else:
        raise Exception("Message is str, not telebot object")

    global message1

    message1 = message

    if message == None:
        return None

    return message1

@bot.message_handler(content_types=['text'])
def func(message):
    try:
        res =
requests.get(f"http://api.openweathermap.org/data/2.5/forecast?id=524901&appi
d={appid}&lang=ru&units=metric")
        data = res.json()
        if ((not data) or (str(data['cod']) == '404')):
            raise Exception('Page not Found 404')
```

```
    except Exception as e:
        bot.send_message(message.chat.id, text=f"Сервер упал :(\nОшибка:
{e}")
        return
    if message == None:
        return
    if (message.text == "Погода"):
        weather = data['list'][0]['weather'][0]['description'].title()
        bot.send_message(message.chat.id, text=weather)
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        back = types.KeyboardButton("Назад")
        markup.add(back)

    elif (message.text == "Температура"):
        temp = 'Температура ' + str(data['list'][0]['main']['temp']) + '°C'
        feels_temp = 'Ощущается как ' +
str(data['list'][0]['main']['feels_like']) + '°C'
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        bot.send_message(message.chat.id, text=temp + "\n" + feels_temp,
reply_markup=markup)
        back = types.KeyboardButton("Назад")
        markup.add(back)

    elif (message.text == "Назад"):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        button1 = types.KeyboardButton("Погода")
        button2 = types.KeyboardButton("Температура")
        markup.add(button1, button2)
        bot.send_message(message.chat.id, text="Вы вернулись в главное меню,
чем могу помочь?", reply_markup=markup)
    else:
        bot.send_message(message.chat.id, text="Не знаю такого..")


class test_class(unittest.TestCase):
    def test_false_parameters(self):
        with self.assertRaises(Exception) as context:
            start("text")
        self.assertEqual(
            "Message is str, not telebot object",
            str(context.exception)
        )

    def test_zero_parameters(self):
        with self.assertRaises(TypeError) as context:
            start()
        self.assertEqual(
            "start() missing 1 required positional argument: 'message'",
            str(context.exception)
        )

    def test_func(self):
        message1 = None
        self.assertEqual(func(message1), None)

    def test_func_zero_parameters(self):
        with self.assertRaises(TypeError) as context:
            func()
        self.assertEqual(
            "func() missing 1 required positional argument: 'message'",
            str(context.exception)
        )
```

```
bot.polling(none_stop=True)
unittest.main()
```