МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МЭИ»
ИНЖЕНЕРНО-ЭКОНОМИЧЕСКИЙ ИНСТИТУТ

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине «Объектно-ориентированный анализ и программирование»
на тему:
«Свойства ООП»

Работу выполнил:
Студент группы ИЭс-160п-19
Зубков Д. Ю.

Принял:
Преподаватель Овсянникова М. Р.

Москва
2021

Оглавление

Условие задачи

Вариант № 7

Разработать программу с использованием класса объектов.

В дополнение к условиям задания № 4

1. Описать новый класс объектов на основе существующего класса «Класс цветных фигур». Добавить новую характеристику для нового класса – цвет фигуры.
2. Создать массив объектов нового класса. Вывести все объекты класса в табличном виде (полный набор характеристик).
3. Вывести все объекты класса в табличном виде. В таблице показать подмножество всех характеристик класса объектов.
4. Вывести данные (в табличном виде с соответствующим заголовком) обо всех фигурах желтого цвета. Предусмотреть реакцию программы на ситуацию, когда объекты с указанными свойствами в таблице отсутствуют.
5. Решить задачу п.4 с использованием модуля.

Фигура: куб

Вычисляемые параметры: площадь поверхности, объем

Метод решения задачи

Задача будет решаться методом декомпозиции задач на составные части, применением математических расчетов.

Формула для решения задачи:

Площадь поверхности: $12\ a^2$

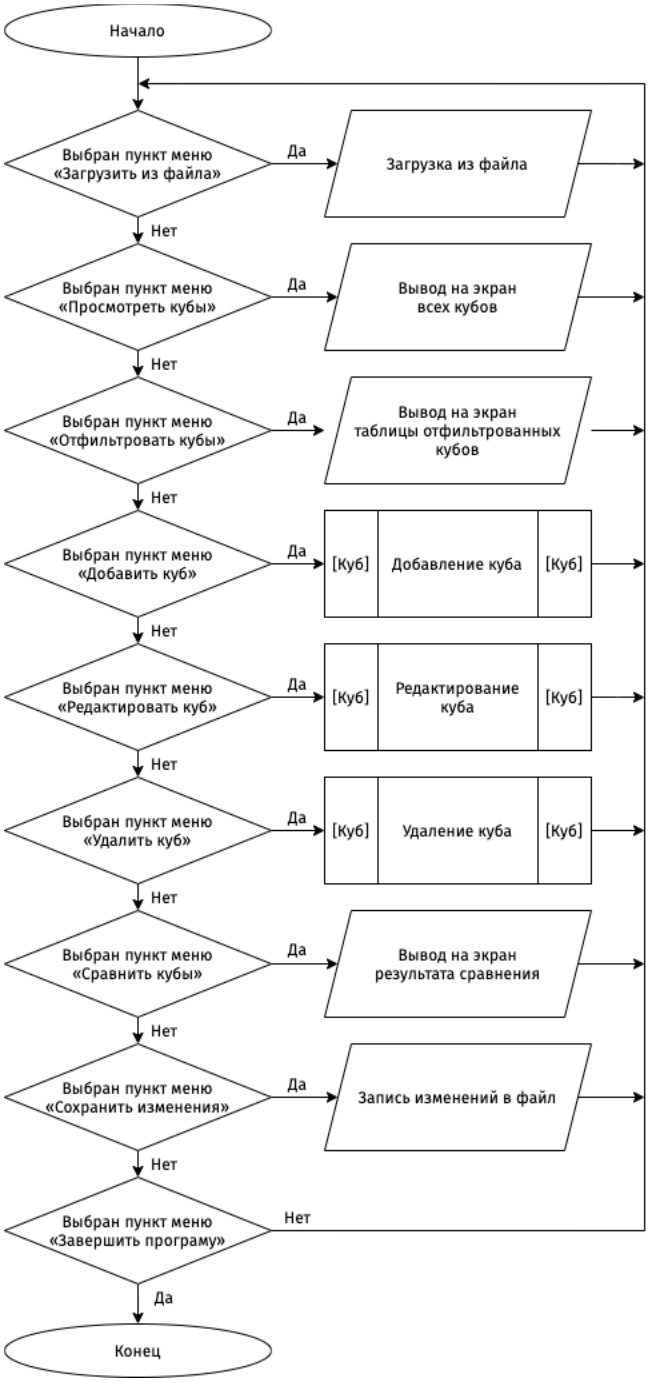
Объем: a^3

Поскольку, справедливо, что при сторонах a и b, где a > b, Sa и Va будут соответственно больше Sb и Vb, сравнение будет проводиться только по стороне куба.

Техническое выполнение задания и тестирование будет проводиться в следующих условиях:

Язык программирования	Python 3.9
Среда разработки	JetBrains PyCharm Community 2020.3
Архитектура	Intel i386 (Core i9 9880H)
Операционная система	Apple macOS 11.2.1

Алгоритм решения задачи



Наборы тестовых данных

Тестовые данные представлены в виде трех .txt файлов.

Файл «cubes.csv»

Тестовые данные	Ожидаемый результат

```
1.12,Yellow
2.45,Green
3.4,Blue
3.1,Yellow
5.0,Black
```

```
№     Side  Perimeter  Area   Volume  Colors
-----------------------------------------------
1     1.12  13.44      7.53   1.40    Yellow
2     3.10  37.20      57.66  29.79   Yellow
```

Состав данных

Класс	Имя	Тип	Структура	Смысл
Входные данные	`cubes`	Класс	Одномерный массив	Исходный массив кубов
Выходные данные	`cubes`	Класс	Одномерный массив	Сохраняемый массив кубов
Промежуточные данные	`cubes_to_compare`	Класс	Одномерный массив	Массив кубов для сравнения
	`filtered_cubes`	Класс	Одномерный массив	Массив отфильтрованных кубов

Код программы

Код модуля «Cube.py»

Класс Cube

```python
class Cube:
    def __init__(self, side):
        self.side = abs(side)

    def __str__(self):
        return f'Side:      {self.side:.2f}\n' \
               f'Perimeter: {self.perimeter:.2f}\n' \
               f'Area:      {self.area:.2f}\n' \
               f'Volume:    {self.volume:.2f}\n'

    def __gt__(self, other):
        return self.side > other.side

    def __eq__(self, other):
        return self.side == other.side

    def __ge__(self, other):
        return self.side >= other.side

    perimeter = property()
    area = property()
    volume = property()

    @perimeter.getter
    def perimeter(self):
        return self.side * 12

    @perimeter.setter
    def perimeter(self, value):
        self.side = value / 12

    @area.getter
    def area(self):
        return self.side ** 2 * 6

    @area.setter
    def area(self, value):
        self.side = (value / 6) ** (1 / float(2))

```

```
    @volume.getter
    def volume(self):
        return self.side ** 3

    @volume.setter
    def volume(self, value):
        self.side = value ** (1 / float(3))

    def present_table(self, number, col_widths):
        columns = [self.side, self.perimeter, self.area, self.volume]
        res_str = ''.join(f'{number:<{col_widths[0]}.0f}')
        for index, param in enumerate(columns):
            if param == int(param):
                res_str += f'{param:<{col_widths[index + 1]}.0f}'
            else:
                res_str += f'{param:<{col_widths[index + 1]}.2f}'
        res_str += '\n'
        return res_str

```

Код модуля «ColoredCube.py»

Класс ColoredCube

```
from Cube import Cube


class ColoredCube(Cube):
    def __init__(self, side, color='Black'):
        Cube.__init__(self, side)
        self.color = color

    def __str__(self):
        return f'Side:      {self.side:.2f}\n' \
               f'Perimeter: {self.perimeter:.2f}\n' \
               f'Area:      {self.area:.2f}\n' \
               f'Volume:    {self.volume:.2f}\n' \
               f'Color:     {self.color}'

    def present_table(self, number, col_widths, show_color=True):
        columns = [self.side, self.perimeter, self.area, self.volume,
self.color]
        res_str = ''.join(f'{number:<{col_widths[0]}.0f}')
        for index, param in enumerate(columns):
            if not show_color and index == 4:
                pass
            else:
                if type(param) is str:
                    res_str += f'{param:<{col_widths[index + 1]}}'
                else:
                    if param == int(param):
                        res_str += f'{param:<{col_widths[index + 1]}.0f}'
                    else:
                        res_str += f'{param:<{col_widths[index + 1]}.2f}'

        res_str += '\n'
        return res_str

```

Код модуля «menu.py»

Классы MenuItem, Menu

```
import os


class MenuItem:
    def __init__(self, text, func, params=None):
        self.text = text
        self.func = func
        self.params = params


class Menu:
    def __init__(self, items: [MenuItem]):
        self.items = items

    def show(self, text='', info_msg='', avoid_clr=False):
        act_n = 0
        while act_n == 0:
            os.system('clear')
            if text != '' and avoid_clr:
                print(text)
            for index, item in enumerate(self.items):
                print(f'{index + 1}: {item.text}')
            if info_msg == '':
                info_msg = '\n'
            print(info_msg)
            usr_inp = input("Type number of action and press return: ")
            try:
                act_n = int(usr_inp)
            except Exception:
                info_msg = f'\nERROR: {usr_inp} -- wrong input'
                act_n = 0

            if act_n > len(self.items) or act_n < 1:
                info_msg = f'\nERROR: {usr_inp} -- wrong input'
                act_n = 0
            else:
                if self.items[act_n - 1].params is not None:
                    act_n, info_msg = self.items[act_n - 1]
.func(*self.items[act_n - 1].params)
                else:
                    act_n, info_msg = self.items[act_n - 1].func()
        if act_n == -1:
            act_n = 0
        return act_n, info_msg

```

Код модуля «io_unit.py»

```
import csv
import os
from ColoredCube import ColoredCube
from menu import Menu, MenuItem


def load_cubes_from_csv(cubes: [ColoredCube]):
    filename = input('Type path to file, or press return: ')
```

```
    os.system('clear')
    if os.path.exists(filename) and os.path.isfile(filename):
        with open(filename, encoding='utf-8') as r_file:
            try:
                file_reader = csv.reader(r_file, delimiter=",")
                new_cubes = []
                for row in file_reader:
                    cube = ColoredCube(float(row[0]), row[1])
                    new_cubes.append(cube)
                if new_cubes:
                    msg = f'\nSuccessfully loaded {len(new_cubes)} cubes'
                else:
                    msg = 'No one cube loaded'
                del cubes[0:len(cubes)]
                cubes += new_cubes
                return 0, msg
            except Exception:
                msg = f'\nERROR: Can’t load cubes, file error'
                return 0, msg
    msg = f'\nERROR: Can’t load cubes, file not exist'
    return 0, msg


def save_to_csv(cubes: [ColoredCube]):
    filename = input('Type name of file: ')
    with open(filename, mode='w', encoding='utf-8') as w_file:
        f_writer = csv.writer(w_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)
        for cube in cubes:
            f_writer.writerow([cube.side, cube.color])
    return 0, f'\nSaved to {filename}'


def get_max_value_length(cubes: [ColoredCube], annotations):
    nums, sides, perimeters, areas, volumes, colors = [], [], [], [], [], []
    for index, cube in enumerate(cubes):
        nums.append(index + 1)
        sides.append(cube.side)
        perimeters.append(cube.perimeter)
        areas.append(cube.area)
        volumes.append(cube.volume)
        colors.append(cube.color)
    table_data = [nums, sides, perimeters, areas, volumes, colors]

    n = [max(5, len(f'{a}  ')) for a in annotations]
    for index, table in enumerate(table_data):
        for item in table:
            if type(item) is str:
                n[index] = max(n[index], len(f'{item}  '))
            else:
                n[index] = max(n[index], len(f'{item:.2f}  '))
    return n


def present_cubes(cubes: [ColoredCube], show_colors=True):
    annotations = ['№', 'Side', 'Perimeter', 'Area', 'Volume', 'Colors']
    col_widths = get_max_value_length(cubes, annotations)
    if not show_colors:
        annotations.pop()
    res_str = ''.join(([f'{a:<{col_widths[index]}}' for index, a in
enumerate(annotations)])) + '\n'
```

```
    res_str += '-' * len(res_str) + '\n'
    for index, cube in enumerate(cubes):
        res_str += cube.present_table(index + 1, col_widths, show_colors)
    return res_str


def show_cubes(cubes: [ColoredCube]):
    text = present_cubes(cubes)
    menu = Menu([
        MenuItem("Back", lambda x, y: (x, y), (-1, ''))
    ])
    return menu.show(avoid_clr=True, text=text)


def filter_cubes(cubes: [ColoredCube]):
    os.system('clear')
    color = input('Type color to filter: ')
    filtered_cubes = list(filter(lambda x: x.color == color,cubes))
    if filtered_cubes:
        text = f'{color} cubes: \n\n' + present_cubes(filtered_cubes,
show_colors=False)
        menu = Menu([
            MenuItem("Back", lambda x, y: (x, y), (-1, ''))
        ])
        return menu.show(avoid_clr=True, text=text)
    else:
        os.system('clear')
        print(f'Array don't have {color} cubes')
        input('Press return')
        msg = ''
        return 0, msg


def add_cubes(cubes: [ColoredCube]):
    os.system('clear')
    num = input('Type number of cubes to add: ')
    msg = ''
    bef_cubes_count = len(cubes)
    try:
        num = int(num)
    except ValueError:
        msg = '\nERROR: can't add cubes, wrong input'
    for i in range(num):
        code, msg = add_cube(cubes, msg)
    cubes_added = len(cubes) - bef_cubes_count
    os.system('clear')
    print(present_cubes(cubes))
    input('Press return')
    msg = f'\nSuccessfully added {cubes_added} cubes'
    return 0, msg


def add_cube(cubes: [ColoredCube], msg=''):
    os.system('clear')
    print(present_cubes(cubes))
    if msg:
        print(msg)
    inp = input('Type side of cube and color divided by space: ')
    try:
        side, color = inp.strip().split(' ')
        side_f = float(side)
```

```
        if side_f > 0:
            cubes.append(ColoredCube(side_f, color))
            msg = f'\nAdded {color} cube with side {side}'
        else:
            msg = '\nERROR: can’t add cube, side must be > 0'
    except ValueError:
        msg = '\nERROR: can’t add cube, wrong input'
    return 0, msg


def edit_cube(cubes: [ColoredCube]):
    os.system('clear')
    print(present_cubes(cubes))
    pos = input('Type № of cube to edit: ')
    try:
        index = int(pos)
        if index - 1 in range(len(cubes)):
            side = input('Type side of cube: ')
            color = input('Type color of cube: ')
            try:
                side_f = float(side)
                if side_f > 0:
                    cubes[index - 1].side = side_f
                    cubes[index - 1].color = color
                    os.system('clear')
                    print(present_cubes(cubes))
                    msg = f'\nSuccessfully edited'
                    input('Press return')
                else:
                    msg = '\nERROR: can’t edit cube, side must be > 0'
            except ValueError:
                msg = '\nERROR: can’t edit cube, wrong input'
        else:
            msg = '\nERROR: can’t edit cube, wrong index'
    except ValueError:
        msg = '\nERROR: can’t edit cube, wrong input'
    return 0, msg


def delete_cube(cubes: [ColoredCube]):
    os.system('clear')
    print(present_cubes(cubes))
    pos = input('Type № of cube to delete: ')
    try:
        index = int(pos)
        if index - 1 in range(len(cubes)):
            cubes.pop(index - 1)
            os.system('clear')
            print(present_cubes(cubes))
            msg = f'\nSuccessfully deleted'
            input('Press return')
        else:
            msg = '\nERROR: can’t edit cube, wrong index'
    except ValueError:
        msg = '\nERROR: can’t edit cube, wrong input'
    return 0, msg


def compare_cubes(cubes: [ColoredCube]):
    os.system('clear')
    print(present_cubes(cubes))
```

```
    pos = input('Type №№ of cubes to compare divided by space: ')
    str_arr = pos.strip().split(' ')
    try:
        index1, index2 = int(str_arr[0]), int(str_arr[1])
        if index1 - 1 in range(len(cubes)) and index2 - 1 in
range(len(cubes)):
            cubes_to_compare = [cubes[index1 - 1], cubes[index2 - 1]]
            os.system('clear')
            menu = Menu([
                MenuItem("Back", lambda x, y: (x, y), (-1, ''))
            ])
            msg = (present_cubes(cubes_to_compare))
            if cubes_to_compare[0] < cubes_to_compare[1]:
                msg += f'\nCube №{index1} less than cube №{index2}'
            elif cubes_to_compare[0] > cubes_to_compare[1]:
                msg += f'\nCube №{index1} greater than cube №{index2}'
            else:
                msg += f'\nCube №{index1} equal cube №{index2}'
            return menu.show(avoid_clr=True, text=msg)
        else:
            msg = '\nERROR: can't compare cubes, wrong index'
    except ValueError:
        msg = '\nERROR: can't compare cubes, wrong input'
    return 0, msg

```

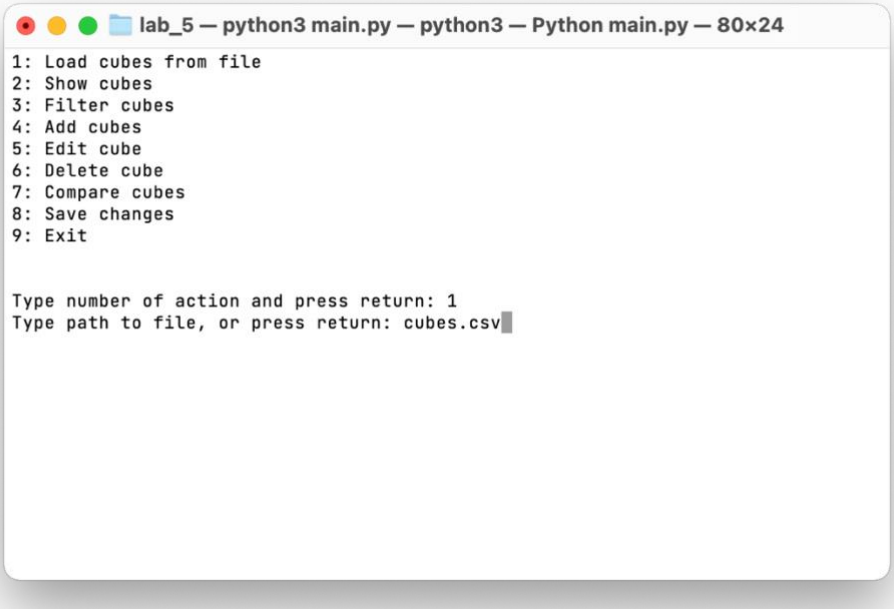
Код исполняемого файла «main.py»

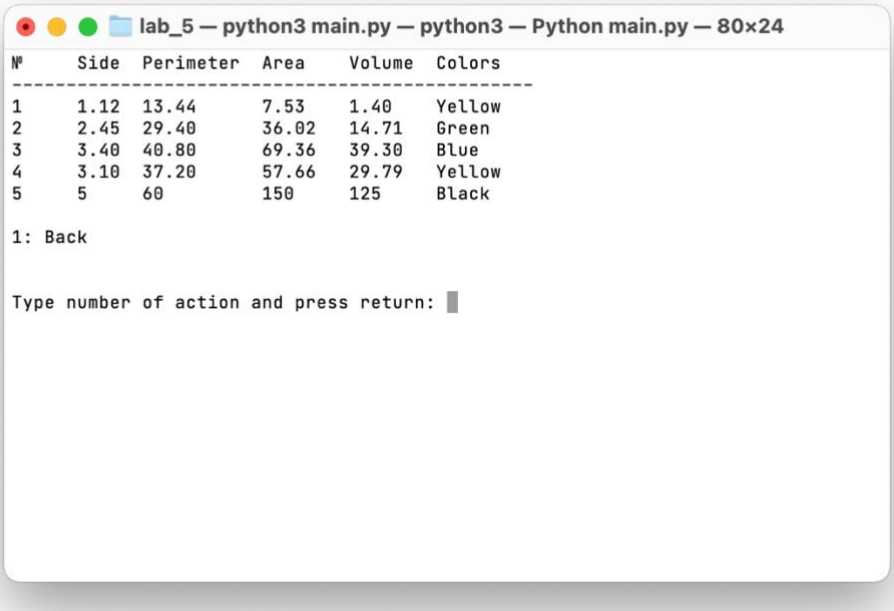
```
from Cube import Cube
from menu import Menu, MenuItem
import io_unit as iou

if __name__ == "__main__":
    cubes: [Cube] = []
    msg = ''

    main_menu = Menu([
        MenuItem('Load cubes from file', iou.load_cubes_from_csv, (cubes, )),
        MenuItem('Show cubes', iou.show_cubes, (cubes,)),
        MenuItem('Filter cubes', iou.filter_cubes, (cubes,)),
        MenuItem('Add cubes', iou.add_cubes, (cubes,)),
        MenuItem('Edit cube', iou.edit_cube, (cubes, )),
        MenuItem('Delete cube', iou.delete_cube, (cubes,)),
        MenuItem('Compare cubes', iou.compare_cubes, (cubes,)),
        MenuItem('Save changes', iou.save_to_csv, (cubes, )),
        MenuItem('Exit', exit, (0,))
    ])
    main_menu.show(avoid_clr=True, info_msg=f'\n{msg}')

```

Тестирование и отладка

1. Загрузка из файла и отображение в табличном виде

Ожидаемый результат: добавление трех кубов в массив и их отображение в табличном виде.

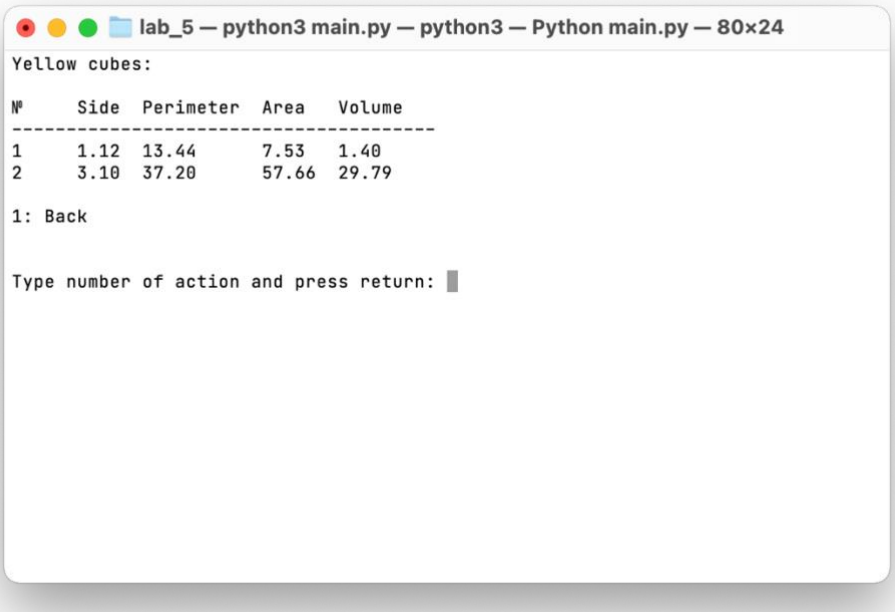




Вывод: кубы загружены в массив из файла и отображаются в табличном виде. Тест пройден.

2. Вывод всех кубов желтого цвета

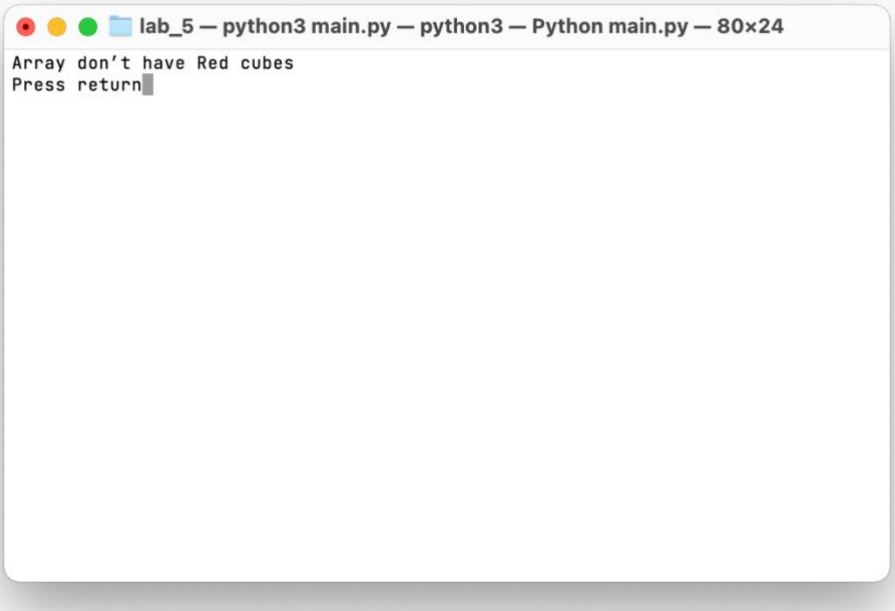
Ожидаемый результат: отображение отфильтрованных кубов.



Вывод: тест пройден.

3. Вывод все кубов красного цвета

Ожидаемый результат: сообщение об ошибке.



Вывод: тест пройден.