

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Избранные главы информатики

ОТЧЕТ
к лабораторной работе
на тему

Работа с файлами, классами, сериализаторами, регулярными выражениями
и стандартными библиотеками

БГУИР КП 1-40 04 01

Выполнила: студентка группы 253502,
Меликова Камилла Раидовна

Проверила: Жвакина Анна Васильевна

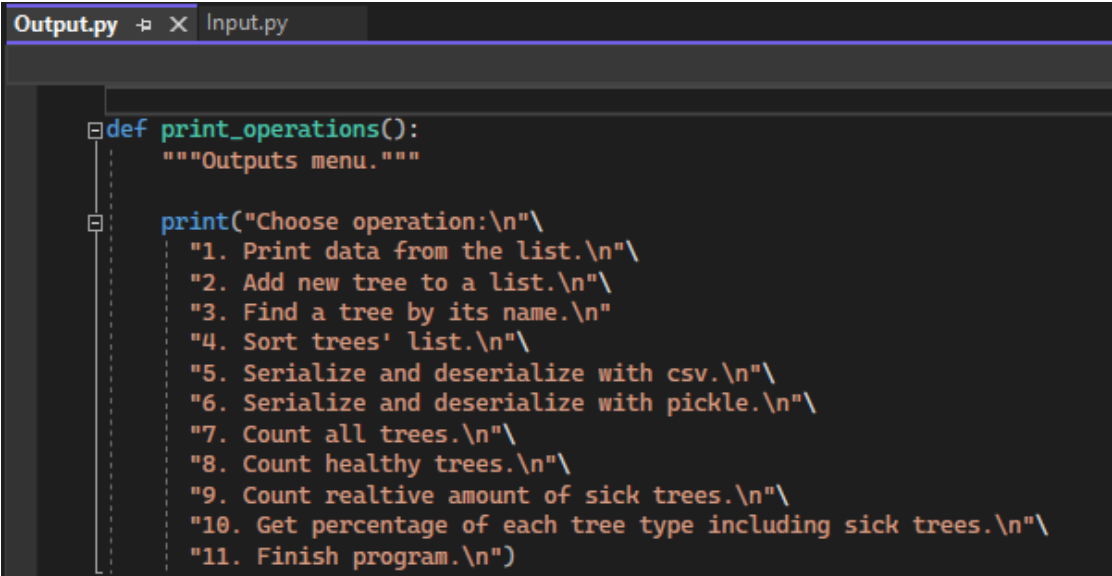
Вариант 14.

Задание 1.

Исходные данные представляют собой словарь. Необходимо поместить их в файл, используя сериализатор. Организовать считывание данных, поиск, сортировку в соответствии с индивидуальным заданием.

Обязательно использовать классы. Реализуйте два варианта: 1) формат файлов CSV; 2) модуль pickle

Хранятся сведения о лесе: вид дерева, общая численность, численность здоровых деревьев. Составьте программу вычисления: 1) суммарного числа деревьев на контрольном участке; 2) суммарного числа здоровых деревьев; 3) относительную численность (%) больных деревьев; 4) относительную численность (%) различных видов, в том числе больных (%) для каждого вида. Выведите информацию о виде дерева, введенном с клавиатуры



```
def print_operations():
    """Outputs menu."""

    print("Choose operation:\n"
          "1. Print data from the list.\n"
          "2. Add new tree to a list.\n"
          "3. Find a tree by its name.\n"
          "4. Sort trees' list.\n"
          "5. Serialize and deserialize with csv.\n"
          "6. Serialize and deserialize with pickle.\n"
          "7. Count all trees.\n"
          "8. Count healthy trees.\n"
          "9. Count realtive amount of sick trees.\n"
          "10. Get percentage of each tree type including sick trees.\n"
          "11. Finish program.\n")
```

Input.py

```
def input_int():
    """Returns integer value."""
    while True:
        try:
            x = int(input("Enter the number:"))
            break;
        except ValueError:
            print("Incorrect input, try again!")
    return x

def check_operation_choice(choice):
    """Checks operation code."""
    if choice not in range(1,12):
        print("Incorrect input, try again.")
        return False
    else:
        return choice

def input_tree_data(trees):
    """Checks input data for tree."""
    print("Please, input following values:")

    print("Input tree type:")
    while True:
        tree_type=str(input())
        for tree in trees:
            if tree['tree_type'] == tree_type:
                print("Tree type already exists in the list,try again.")
                break
        else:
            break

    print("Input trees' amount:")
    while True:
        amount=input_int()
        if amount<=0:
            print("Incorrect input,try again.")
        else:
            break

    print("Input healthy trees' amount:")
    while True:
        healthy_amount=input_int()
        if healthy_amount<=0 or healthy_amount>amount:
            print("Incorrect input,try again.")
        else:
            break

    return {'tree_type':tree_type,'total_count':amount,'healthy_count':healthy_amount}
```

```

import csv
import pickle

class TreeFileManager(object):
    @staticmethod
    def csv_serializer(filename,trees):
        """Writes given data to a csv file using DictWriter."""

        with open(filename,'w',newline='') as fh:
            columns=['tree_type', 'total_count', 'healthy_count']
            writer=csv.DictWriter(fh,fieldnames=columns)
            writer.writeheader()
            for tree in trees:
                writer.writerow(tree)
            print("Data is serialized with CSV.")

    @staticmethod
    def csv_deserializer(filename):
        """Loads and returns data from csv file using DictReader."""

        trees=[]
        with open(filename,'r',newline='') as fh:
            reader=csv.DictReader(fh)
            trees=list(reader)
        return trees

    @staticmethod
    def pickle_serializer(filename,trees):
        """Writes given data to a file using pickle."""

        with open(filename,'wb') as fh:
            pickle.dump(trees,fh)
            print("Data is serialized with pickle.")

    @staticmethod
    def pickle_deserializer(filename):
        """Reads data from a file and returns the object."""

        with open(filename,'rb') as fh:
            trees=pickle.load(fh)
        return trees

    @staticmethod
    def print_data(trees):
        """ Prints deserialized data."""

        print("Deserialized data:")
        for tree in trees:
            print(tree)
        print("\n")

```

```

class Tree(object):

    def __init__(self, trees):
        """Initializes class object with a list of dictionaries."""

        self.trees = trees

    def add_tree(self, new_tree):
        """Add new tree to the list."""

        self.trees.append(new_tree)

    def count_all_trees(self):
        """Counts amount of all trees."""

        return sum(tree['total_count'] for tree in self.trees)

    def count_healthy_trees(self):
        """Counts amount of healthy trees."""

        return sum(tree['healthy_count'] for tree in self.trees)

    def count_sick_trees(self):
        """Counts amount of sick trees."""

        return self.count_all_trees()-self.count_healthy_trees()

    def count_relative_sick_trees(self):
        """Counts relative amount of sick trees."""

        return round(self.count_sick_trees()/self.count_all_trees()*100)

    def get_relative_data(self):
        """Counts relative amount of each type including sick trees."""

        data=[]
        for tree in self.trees:
            relative_count=round(tree['total_count']/self.count_all_trees()*100)
            relative_sick_count=round((tree['total_count']-tree['healthy_count']/self.count_all_trees()*100)
            statistics={
                'tree_type':tree['tree_type'],
                'relative_count':relative_count,
                'relative_sick_count':relative_sick_count
            }
            data.append((statistics))
        return data

```

```

def print_tree_data(self, tree):
    """Prints all data about a specific tree."""

    print('Tree type:', tree['tree_type'])
    print('Amount', tree['total_count'])
    print('Healthy amount', tree['healthy_count'])
    print('\n')

def find_tree(self, tree_type):
    """Searches for a tree in the list of trees."""

    for tree in self.trees:
        if tree['tree_type'] == tree_type:
            print("Tree found:")
            self.print_tree_data(tree)
            return tree
    print("Tree not found.\n")
    return None

def sort_trees(self):
    """Sorts given list."""

    self.trees.sort(key=lambda x: x['tree_type'])
    print("List is sorted.\n")

```

This program performs operations on trees' list, serializes and deserializes its data.
Choose operation:

1. Print data from the list.
2. Add new tree to a list.
3. Find a tree by its name.
4. Sort trees' list.
5. Serialize and deserialize with csv.
6. Serialize and deserialize with pickle.
7. Count all trees.
8. Count healthy trees.
9. Count realtive amount of sick trees.
10. Get percentage of each tree type including sick trees.
11. Finish program.

Enter the number:1

Tree type: Oak

Amount 100

Healthy amount 80

Tree type: Pine

Amount 150

Healthy amount 120

Tree type: Maple

Amount 81

Healthy amount 60

Задание 2. В соответствии с заданием своего варианта составить программу для анализа текста. Считать из исходного файла текст. Используя регулярные выражения получить искомую информацию (см. условие), вывести ее на экран и сохранить в другой файл. Заархивировать файл с результатом с помощью модуля zipfile и обеспечить получение информации о файле в архиве.

Также выполнить общее задание – определить и сохранить в файл с результатами:

- количество предложений в тексте;
- количество предложений в тексте каждого вида отдельно (повествовательные, вопросительные и побудительные);
- среднюю длину предложения в символах (считаются только слова);
- среднюю длину слова в тексте в символах;
- количество смайликов в заданном тексте. Смайликом будем считать последовательность символов, удовлетворяющую условиям:
 - первым символом является либо «;» (точка с запятой) либо «:» (двоеточие) ровно один раз;
 - далее может идти символ «-» (минус) сколько угодно раз (в том числе символ минус может идти ноль раз);
 - в конце обязательно идет некоторое количество (не меньше одной) одинаковых скобок из следующего набора: «(», «)», «[», «]»;
 - внутри смайлика не может встречаться никаких других символов.Например, эта последовательность является смайликом: «;-----[[[[[[[». Эти последовательности смайликами не являются: «]», «;--», «:», «)».

Получить список дат (формат 2007)

Из заданной строки получить список слов, у которых третья с конца буква согласная, а предпоследняя – гласная.

определить число слов в строке, начинающихся с гласной;

найти слова, содержащие две одинаковые буквы подряд и их порядковые номера;

вывести слова в алфавитном порядке.

```
import zipfile

class FileManager:
    @staticmethod
    def read_from_file(filename):
        with open(filename, "r") as file:
            res = file.read()
            print(res)
            return res

    @staticmethod
    def write_to_file(filename, data):
        with open(filename, 'w') as file:
            for item in data:
                file.write(str(item) + '\n')

    @staticmethod
    def archive(filename):
        with zipfile.ZipFile('output.zip', 'w') as zipf:
            zipf.write(filename)

    @staticmethod
    def get_info_about_file():
        with zipfile.ZipFile('output.zip', 'r') as zipf:
            file_infos = zipf.infolist()

            for file_info in file_infos:
                print("Имя файла:", file_info.filename)
                print("Размер файла (байт):", file_info.file_size)
                print("Размер сжатого файла (байт):", file_info.compress_size)
                print("Метод сжатия:", file_info.compress_type)
                print("Дата и время модификации:", file_info.date_time)
                print("CRC32 хэш файла:", file_info.CRC)
                print()
```



```

import re

class TextReader:
    def __init__(self):
        self.text=None

    def count_sentences(self):
        """Counts sentences in a text."""

        sentences=re.findall(r'(?!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?|\!)\s', self.text)
        return len(sentences)

    def count_sentences_by_type(self):
        """Counts each type of sentence: declarative, interrogative, affirmative."""

        declarative_sentences = re.findall(r'(?!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.)\s', self.text)
        interrogative_sentences = re.findall(r'(?!\w\.\w.)(?<![A-Z][a-z]\.)(?<=?)\s', self.text)
        imperative_sentences = re.findall(r'(?!\w\.\w.)(?<![A-Z][a-z]\.)(?<=!)\s', self.text)

        return len(declarative_sentences), len(interrogative_sentences), len(imperative_sentences)

    def average_sentence_length(self):
        """Counts average sentence's length in chars with only words."""

        sentences = re.findall(r'(?!\w\.\w.)(?<![A-Z][a-z]\.)([.!?]\s*(.*?)(?=[.!?])\Z)', self.text, flags=re.DOTALL)
        total_length = 0

        for sentence in sentences:
            words = re.findall(r'\b\w+\b', sentence)
            total_length += sum(len(word) for word in words)

        average_length = total_length / len(sentences)
        return average_length

    def average_word_length(self):
        """Counts average word's length in chars."""

        words = re.findall(r'\b\w+\b', self.text)
        total_length = 0

        for word in words:
            total_length += len(word)

        average_length = total_length / len(words)
        return average_length

```

```

class Analyzer(TextReader,FileManager):
    def __init__(self):
        super().__init__()

    def find_dates(self):
        """Returns dates in format dd.mm.yyyy"""

        #pattern = r'\b\d{4}\b' for only year
        dates = re.findall(r'\d{2}\.\d{2}\.\d{4}',self.text)
        return dates

    def count_specific_words(self,text):
        """Returns words from a line with a vowel at n-1 position and consonant at n-2 position."""

        words = re.findall(r'\b\w+\b', text)
        selected_words=[]
        pattern = r'.*[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ][aeiouyAEIOUY].$'
        for word in words:
            if re.match(pattern, word):
                selected_words.append(word)
        return selected_words

    def count_vowel_begin_words(self,text):
        """Returns vowel beginning words from a line."""

        words = re.findall(r'\b\w+\b', text)
        selected_words=[]
        pattern = r'\b[aeiouyAEIOUY][a-zA-Z]*\b'
        for word in words:
            if re.match(pattern, word):
                selected_words.append(word)
        return selected_words

    def find_words_with_double_letters(self):
        """Returns words with double letters, their index and the double letter itself."""

        words = re.findall(r'\b\w+\b', self.text)
        #pattern = r'\b(\w*(\w)\2\w*)\b'
        pattern=r'\b([a-zA-Z]*(\w)\2[a-zA-Z]*)\b'

        matches = re.finditer(pattern, self.text)

        results = []

        for match in matches:
            word = match.group(1)
            letter = match.group(2)
            index = words.index(word)
            results.append((word, index, letter))

```

```
result.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Количество предложений: 5
Декларативные предложения: 3
Интеррогативные предложения: 1
Побудительные предложения: 1
Средняя длина предложения: 17.166666666666668
Средняя длина слова: 3.7567567567567566
Количество смайликов: 2
Даты:
12.04.2007
Выбранные слова:
пор
Слова, начинающиеся с гласной:
Ahaha
you
your
Слова с двумя одинаковыми буквами подряд:
occurred: 9:r
sunny: 13:n
Слова в алфавитном порядке:
04
12
2007
Alice
Definitely
I
In
It
May
Yes
a
a
an
character
curious
```

Задание 3.В соответствии с заданием своего варианта доработать программу из ЛР3, используя класс и обеспечить:

а) определение дополнительных параметров среднее арифметическое элементов последовательности, медиана, мода, дисперсия, СКО последовательности;

б) с помощью библиотеки matplotlib нарисовать графики разных цветов в одной координатной оси:

- график по полученным данным разложения функции в ряд, представленным в таблице,
- график соответствующей функции, представленной с помощью модуля math. Обеспечить отображение координатных осей, легенды, текста и аннотации.

14.

$$\ln(1-x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{n} = -x - \frac{x^2}{2} - \frac{x^3}{3} + \dots, |x| < 1$$

```

class Calculator:
    instance_count = 0    # Static attribute which counts amount of class objects.

    def __init__(self):    # Initialize class with two dynamic attributes: x and sequence of components.
        Calculator.instance_count+=1
        self._x = None
        self.seq=[]

    @property
    def x(self):           # Get value of argument x.
        return self._x

    @x.setter             # Set value to argument x.
    def x(self, value):
        self._x = value

    def get_function_value(self):
        """Calculates the function ln(1-x).

        Arguments:
        x -- float argument, |x|<1
        """
        return math.log(1-self.x)

    def get_series_sum(self,eps,y):
        """Calculates the sum of given series.

        Arguments:
        x -- float argument
        eps -- specified accuracy
        y -- function's value

        Returns the sum and amount of iterations,
        alerts, if accuracy hasn't been achieved.
        """

        sum,component,n,prev_result=0,0,0,1000
        while True:
            n+=1
            component=-1*self.x**n/n
            sum+=component
            self.seq.append(sum)
            if n==500:
                print("500 iterations already done.")
                return sum,n;
            if(abs(sum-y)<eps):
                return sum,n
            elif prev_result<=abs(sum-y):
                print("Defined accuracy can't be achieved.")
                return sum,n
            prev_result=abs(sum-y)

```

```

class ImprovedCalculator(Calculator):
    def __init__(self):
        super().__init__()

    def calculate_sequence_mean(self):
        """Calculates sequence's mean value."""

        if self.seq:
            return statistics.mean(self.seq)
        else:
            return None

    def calculate_median(self):
        """Calculates sequence's median value."""

        if self.seq:
            self.seq.sort()
            return statistics.median(self.seq)
        else:
            return None

    def calculate_mode(self):
        """Calculates sequence's mode value."""

        try:
            if self.seq:
                return statistics.mode(self.seq)
            else:
                return None
        except statistics.StatisticsError:
            return None

    def calculate_variance(self):
        """Calculates sequence's variance value."""

        if self.seq:
            return statistics.variance(self.seq)
        else:
            return None

    def calculate_stdev(self):
        """Calculates sequence's root mean square deviation value."""

        if self.seq:
            return statistics.stdev(self.seq)
        else:
            return None

```

```

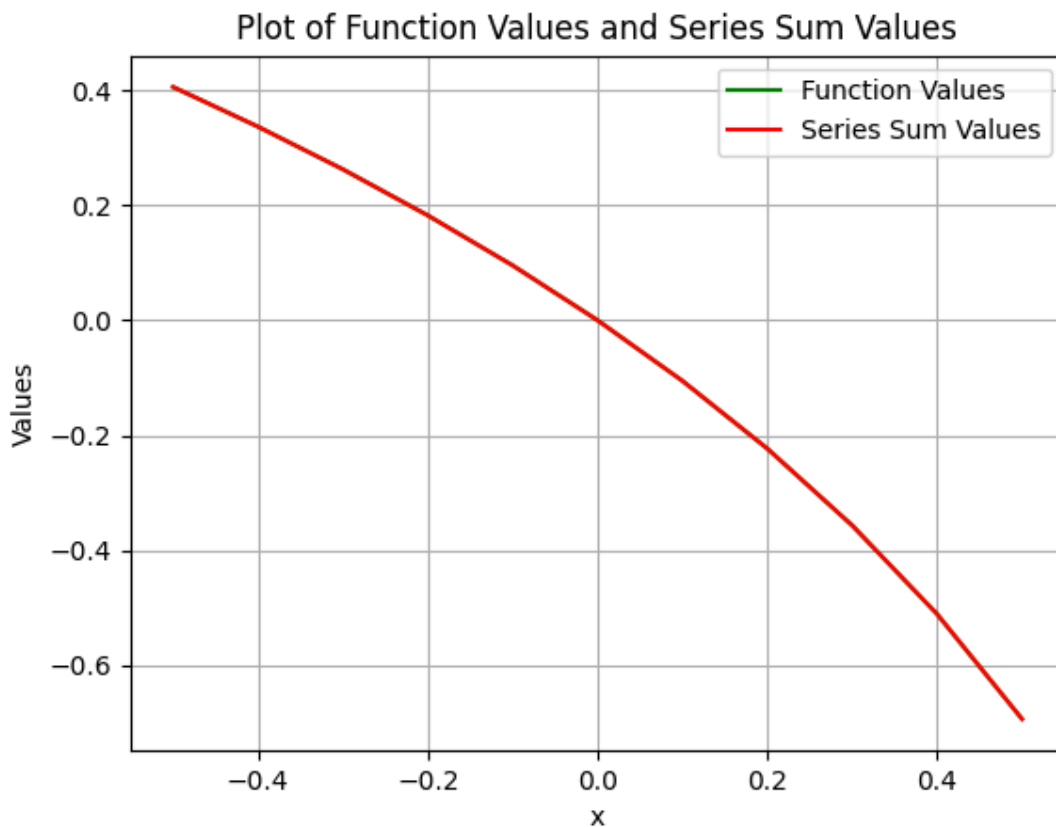
class PlotManager():
    @staticmethod
    def plot_data(x_values, function_values, series_values,filename):
        """Builds plots of function and series values and saves them to file."""

        plt.plot(x_values, function_values, label='Function Values',color='green')
        plt.plot(x_values, series_values, label='Series Sum Values',color='red')
        plt.xlabel('x')
        plt.ylabel('Values')
        plt.title('Plot of Function Values and Series Sum Values')
        plt.legend()
        plt.grid(True)

        plt.savefig(filename)
        print(f"Plot is saved here: {filename}")

        plt.show()

```



Задание 4.

Построить квадрат, описанный около окружности с радиусом R .

```

from abc import ABC, abstractmethod

class GeometricShape(ABC):
    @abstractmethod
    def calculate_area(self):
        ...

    @abstractmethod
    def draw(self):
        ...

```

```

class ShapeColor:
    def __init__(self, color):
        self._color = color

    @property
    def color(self):
        return self._color

    @color.setter
    def color(self, value):
        self._color = value

```

```

class Circle(GeometricShape):
    def __init__(self, radius, color, name):
        self.radius = radius
        self.shape_color = ShapeColor(color)
        self.name = name

    def calculate_area(self):
        """Calculates circle's area."""
        return math.pi * self.radius ** 2

    def get_shape_parameters(self):
        """Returns circle's parameters."""
        print( "Radius: {}\nColor: {}\nArea: {}\nName: {}\n" \
              .format(self.radius, self.shape_color.color, self.calculate_area(), self.name))

    def draw(self, axes):
        """Draws a circle with given data."""
        Drawing_colored_circle = plt.Circle((self.radius, self.radius), self.radius, color=self.shape_color.color)
        axes.add_artist(Drawing_colored_circle)

```

```

class Square(GeometricShape):
    def __init__(self,side,color,name):
        self.side=side
        self.shape_color=ShapeColor(color)
        self.name=name

    def calculate_area(self):
        """Calculates circle's area."""

        return self.side ** 2

    def get_shape_parameters(self):
        """Returns square's parameters."""

        print( "Side: {}\nColor: {}\nArea: {}\nName: {}\n"\
            .format(self.side, self.shape_color.color,self.calculate_area(),self.name))

    def draw(self,axes):
        """Draws a square with given data."""

        axes.fill([0, self.side, self.side, 0], [0, 0, self.side, self.side], color=self.shape_color.color)

```

```

import __main__

from Square import Square
from Circle import Circle
import matplotlib.pyplot as plt
import Input as inp

def get_square():
    """Returns an object of Square."""

    print("Enter square side:")
    side = inp.input_float()
    s_color = inp.input_color()
    s_name=str(input("Enter square name:"))
    print("")
    square = Square(side,s_color,s_name)
    return square

def get_circle(square):
    """Returns an object of Circle."""

    c_color=inp.input_color()
    c_name=str(input("Name:"))
    print("")
    circle=Circle((square.side)/2,c_color,c_name)
    return circle

def build_plot(square,circle,filename):
    """Builds both square and circle and saves image to file."""

    _, ax = plt.subplots()
    circle.draw(ax)
    square.draw(ax)

    ax.set_aspect('equal', adjustable='box')
    ax.plot([], [], color=square.shape_color.color, label=square.name)
    ax.plot([], [], color=circle.shape_color.color, label=circle.name)
    ax.legend()

    plt.savefig(filename)
    plt.show()

```



```

import sys
import Task4

def main():
    while True:
        Task4.inp.print_all_colors()

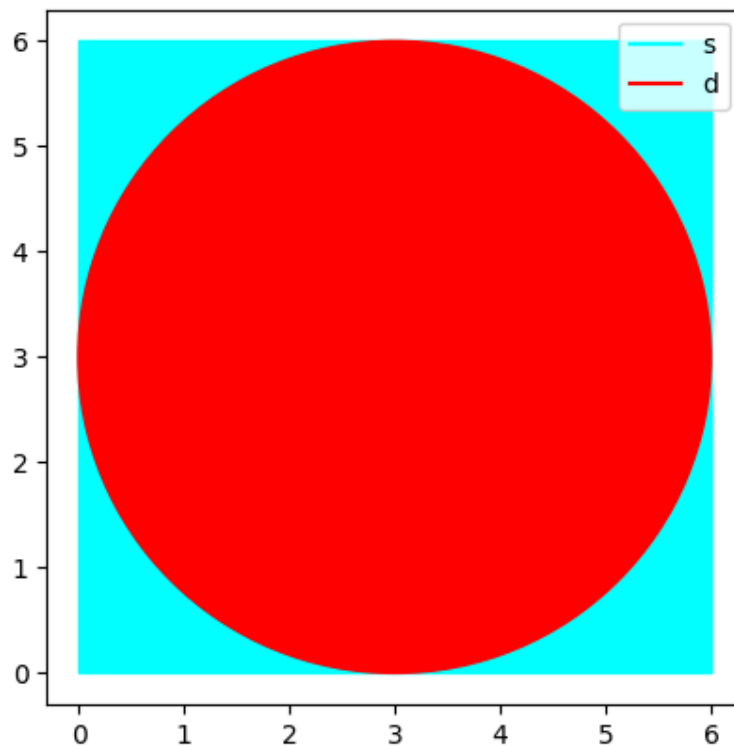
        square=Task4.get_square()
        circle=Task4.get_circle(square)

        square.get_shape_parameters()
        circle.get_shape_parameters()
        Task4.build_plot(square,circle,'plots.png')

        if not Task4.inp.continue_or_exit():
            break

if __name__ == "__main__":
    print("We are here.")
    sys.exit(main())

```



Задание 5. Вычислить сумму элементов матрицы, расположенных ниже главной диагонали. Вычислить стандартное отклонение для элементов главной диагонали матрицы. Ответ округлите до сотых. Вычисление стандартного отклонения выполнить двумя способами: через стандартную функцию и через программирование формулы.

```
class Matrix:
    def __init__(self, rows, cols):
        """Initializes matrix with random values from 0 to 10."""
        self._matrix = np.array([[random.randint(0, 10) for _ in range(cols)] for _ in range(rows)])
        self.rows = rows
        self.cols = cols

    @property
    def matrix(self):
        return self._matrix

    @matrix.setter
    def matrix(self, value):
        self._matrix = value

    def print_matrix(self):
        """Prints matrix."""
        for row in self.matrix:
            for elem in row:
                print(elem, end=" ")
            print("")

    def __add__(self, other):
        print("I can't calculate sum, but my son can.")
```

```
class MatrixOperation(Matrix):
    def __init__(self, rows, cols):
        super().__init__(rows, cols)

    def __add__(self, other):
        """Magic method, returns sum of matrixes."""
        if self.matrix.shape != other.matrix.shape:
            raise ValueError("Matrices must have the same dimensions to perform addition.")

        result = MatrixOperation(self.matrix.shape[0], self.matrix.shape[1])
        result.matrix = self.matrix + other.matrix
        return result

    def __sub__(self, other):
        """Magic method, returns difference between matrixes."""
        if self.matrix.shape != other.matrix.shape:
            raise ValueError("Matrices must have the same dimensions to perform subtraction.")

        result = MatrixOperation(self.matrix.shape[0], self.matrix.shape[1])
        result.matrix = self.matrix - other.matrix
        return result

    def __mul__(self, other):
        """Magic method, multiplies two matrixes."""
        if self.cols != other.rows:
            raise ValueError("Number of columns of the first matrix must be equal to the number of rows of the second matrix for multiplication.")

        result = MatrixOperation(self.rows, other.cols)
        for i in range(self.rows):
            for j in range(other.cols):
                for k in range(self.cols):
                    result.matrix[i][j] += self.matrix[i][k] * other.matrix[k][j]
        return result
```

```

def sum_below_diagonal(self):
    """Counts sum of elements below main diagonal."""

    rows = len(self.matrix)
    cols = len(self.matrix[0])
    total = 0
    for i in range(rows):
        for j in range(cols):
            if i > j:
                total += self.matrix[i][j]
    return total

def get_main_diag_elements(self):
    """Returns main diagonal elements."""

    elements=[]
    rows = len(self.matrix)
    cols = len(self.matrix[0])
    for i in range(rows):
        for j in range(cols):
            if i == j:
                elements.append(self.matrix[i][j])
    return elements

def math_square_deviation(self):
    """Returns square deviation for main diagonal elements,built-in method."""

    elements=self.get_main_diag_elements()
    return round(np.std(elements),2)

def manual_square_deviation(self):
    """Returns square deviation for main diagonal elements,formula."""

    elements=self.get_main_diag_elements()
    mean_value=statistics.mean(elements)
    num_sum=0
    for elem in elements:
        num_sum+=(elem-mean_value)**2

    return round(math.sqrt(num_sum/len(elements)),2)

```

```

from MatrixOperation import MatrixOperation

matrix_1=MatrixOperation(5,5)
matrix_2=MatrixOperation(5,5)

matrix_1.print_matrix()
print("")
matrix_2.print_matrix()

sum_matrix=matrix_1+matrix_2
print("\nSum:")
sum_matrix.print_matrix()

diff_matrix=matrix_1-matrix_2
print("\nDifference:")
diff_matrix.print_matrix()

mul_matrix=matrix_1*matrix_2
print("\nMultiplication:")
mul_matrix.print_matrix()

print("\nSum of elements below main diagonal for sum matrix:",sum_matrix.sum_below_diagonal())
print("\nSquare deviation for main diagonal elements (built-in) for sum matrix:",sum_matrix.math_square_deviation())
print("\nSquare deviation for main diagonal elements (formula) for sum matrix:",sum_matrix.manual_square_deviation())

```

```
8 3 2 5 9
3 3 2 1 7
8 5 3 1 1
3 10 8 2 7
4 9 4 2 6
```

```
4 0 5 0 0
2 2 8 3 0
2 5 6 3 9
6 6 2 3 6
3 1 10 9 1
```

Sum:

```
12 3 7 5 9
5 5 10 4 7
10 10 9 4 10
9 16 10 5 13
7 10 14 11 7
```

Difference:

```
4 3 -3 5 9
1 1 -6 -2 7
6 0 -3 -2 -8
-3 4 6 -1 1
1 8 -6 -7 5
```

Multiplication:

```
104 59 183 121 58
55 34 131 89 32
63 34 117 44 37
91 83 219 132 99
76 58 182 108 60
```

Sum of elements below main diagonal for sum matrix: 102

Square deviation for main diagonal elements (built-in) for sum matrix: 2.65