Лабораторная работа №4
Дисциплина “Избранные главы информатики”
Выполнил Антихович М.В., гр. 253504
Вариант 1

Файл main.py

main.py

```python
from task1 import task1 as t1
from task2 import task2 as t2
from task3 import task3 as t3
from task4 import task4 as t4
from task5 import task5 as t5

if __name__ == "__main__":
    while True:
        input_choice = int(
            input(
                "\n---Choose task---\n"
                "-1: Task1\n"
                "-2: Task2\n"
                "-3: Task3\n"
                "-4: Task4\n"
                "-5: Task5\n"
                "-0: Exit\n"
            )
        )
        match input_choice:
            case 0:
                print("---Bye bye---1")
                break
            case 1:
                t1.task1_run()
            case 2:
                t2.task2_run()
            case 3:
                t3.task3_run()
            case 4:
                t4.task4_run()
            case 5:
                t5.task5_run()
```

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

В сводке об экспортируемых товарах указывается: наименование товара, страна, импортирующая товар, объем поставляемой партии в штуках.

Напечатайте списки стран, в которые экспортируется данный товар, и общий объем его экспорта. Выведите информацию о товаре, введенном с клавиатуры

Класс Good

```python
class Good:
    def __init__(self, name, exporting_country, count):
        self.name = name
        self.exporting_country = exporting_country
        self.count = count
```

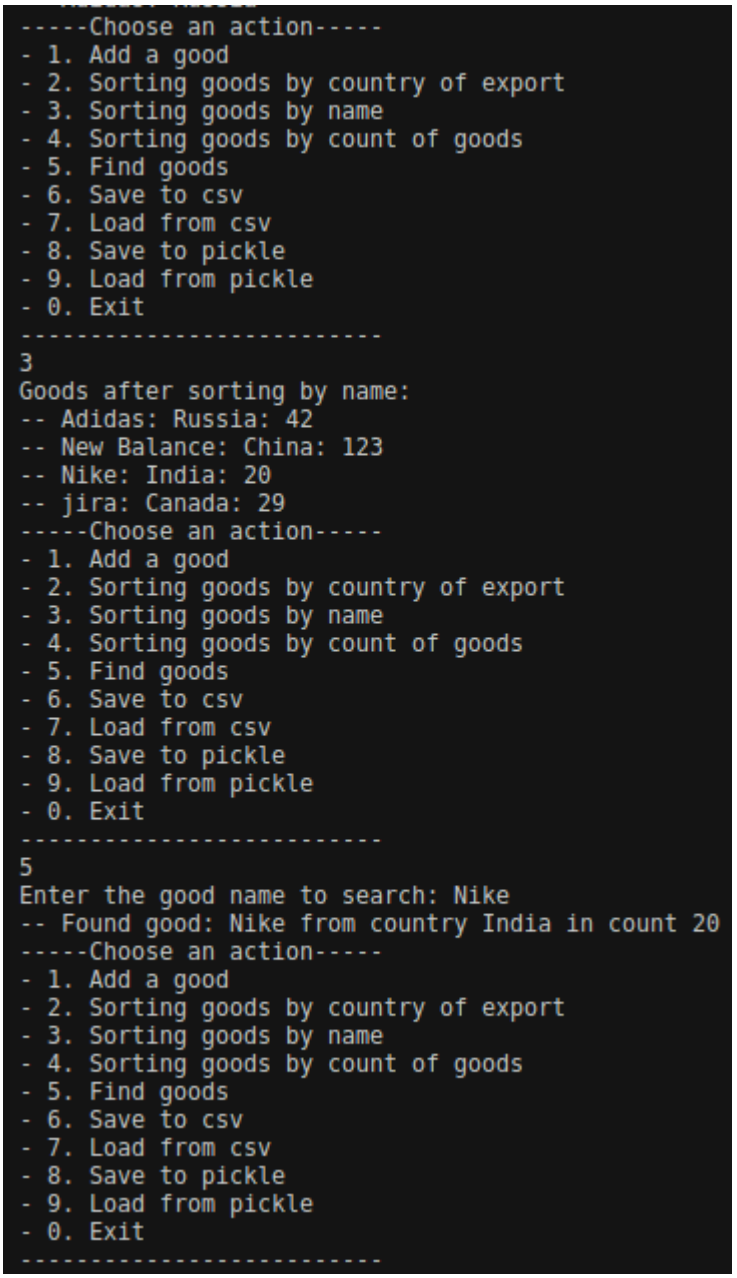
Класс GoodSummary

```python
class GoodSummary:
    def __init__(self):
        self.goods = []

    def add_good(self, good):
        self.goods.append(good)

    def find_good(self, good_name):
        for good in self.goods:
            if good.name == good_name:
                return good
        return None

    def sort_goods(self, key="name"):
        if key == "name":
            self.goods.sort(key=lambda x: x.name)
        elif key == "country":
            self.goods.sort(key=lambda x: x.exporting_country)
        elif key == "count":
            self.goods.sort(key=lambda x: x.count)
        else:
            print("Invalid sort key")

    def find_countries_for_product(self, good_name):
        countries = set()
        total_count = 0
        for good in self.goods:
            if good.name == good_name:
                countries.add(good.exporting_country)
                total_count += good.count
        return list(countries), total_count

    def save_to_csv(self, filename):
        with open(filename, "w") as csv_file:
            writer = csv.writer(csv_file)
            writer.writerow(["Name", "Countries", "Count"])
            for good in self.goods:
                writer.writerow([good.name, good.exporting_country, good.count])
```

```python
    @staticmethod
    def load_from_csv(filename):
        good_summary = GoodSummary()
        with open(filename, "r") as csv_file:
            reader = csv.reader(csv_file)
            next(reader)
            for row in reader:
                name, exporting_country, count = row
                good = Good(name, exporting_country, count)
                good_summary.add_good(good)
        return good_summary

    def save_to_pickle(self, filename):
        with open(filename, "wb") as pickle_file:
            pickle.dump(self.goods, pickle_file)

    @staticmethod
    def load_from_pickle(filename):
        good_summary = GoodSummary()
        with open(filename, "rb") as pickle_file:
            good_summary.goods = pickle.load(pickle_file)
        return good_summary
```

Результаты выполнения

```
-----Choose an action-----
- 1. Add a good
- 2. Sorting goods by country of export
- 3. Sorting goods by name
- 4. Sorting goods by count of goods
- 5. Find goods
- 6. Save to csv
- 7. Load from csv
- 8. Save to pickle
- 9. Load from pickle
- 0. Exit
--------------------------
7
New Balance China 123
Nike India 20
jira Canada 29
Adidas Russia 42
-----Choose an action-----
- 1. Add a good
- 2. Sorting goods by country of export
- 3. Sorting goods by name
- 4. Sorting goods by count of goods
- 5. Find goods
- 6. Save to csv
- 7. Load from csv
- 8. Save to pickle
- 9. Load from pickle
- 0. Exit
--------------------------
2
Goods after sorting by export country:
-- jira: Canada
-- New Balance: China
-- Nike: India
-- Adidas: Russia
```



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.

Вывести все заглавные английские буквы

В заданном тексте заменить последовательность символов «a…ab…bc…c» (букв a и c в последовательности больше 0, букв b – больше единицы) на последовательность «qqq».

определить, сколько слов имеют максимальную длину;

вывести все слова, за которыми следует запятая или точка;

найти самое длинное слово, которое заканчивается на 'e'

Класс TextAnalyzer

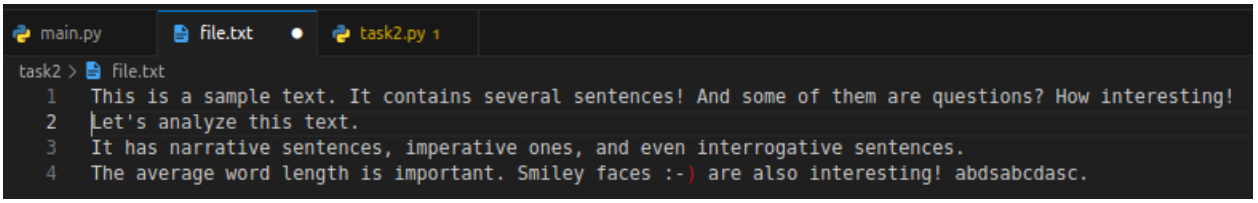
```python
class TextAnalyzer:
    def __init__(self, filename):
        self.filename = filename
        self.text = self.read_text_from_file()

    def read_text_from_file(self):
        with open(self.filename, "r", encoding="utf-8") as file:
            return file.read()

    def analyze_text(self):
        text = self.text
        # Подсчет количества предложений каждого вида
        sentences = re.split(r"[.!?]", text)
        sentences_count = len(sentences) - 1  # учтем пустую строку в конце списка
        narr_sentences = 0
        interrogative_sentences = 0
        imperative_sentences = 0
        (variable) narr_sentences_count: int
                                                *\.+', text)
        narr_sentences_count = len(narr_sentences)
        imperative_sentences = re.findall(r'[^\.\?!]*!+', text)
        imperative_sentences_count = len(imperative_sentences)
        interrogative_sentences =  re.findall(r'[^\.\?!]*\?+', text)
        interrogative_sentences_count = len(interrogative_sentences)

        # Средняя длина предложения в символах
        total_sentence_length = sum(len(sentence.split()) for sentence in sentences)
        average_sentence_length = total_sentence_length / sentences_count

        # Средняя длина слова в символах
        words = re.findall(r"\b\w+\b", text)
        total_word_length = sum(len(word) for word in words)
        average_word_length = total_word_length / len(words)

        # Поиск смайликов
        smiles = re.findall(r"[:;]-*[\(\[\)\]]+", text)
        num_smiles = len(smiles)

        return (
            sentences_count,
            narr_sentences_count,
            interrogative_sentences_count,
            imperative_sentences_count,
            average_sentence_length,
            average_word_length,
            num_smiles,
        )
```

```python
    def find_all_caps(self):
        text = self.text
        all_caps = re.findall(r"[A-Z]", text)
        return all_caps

    def replace_sequence(self):
        text = self.text
        replaced_text = re.sub(r"(a+b+c+)", "qqq", text)
        return replaced_text

    def max_length_words(self):
        text = self.text
        words = re.findall(r"\b\w+\b", text)
        max_length = max(len(word) for word in words)
        max_length_words = [word for word in words if len(word) == max_length]
        return len(max_length_words),max_length

    def words_followed_by_punctuation(self):
        text = self.text
        word_punctuation = re.findall(r"\b\w+[.,]", text)
        return word_punctuation

    def longest_word_ending_with_e(self):
        text = self.text
        words = re.findall(r"\b\w+\b", text)
        longest_word = ""
        for word in words:
            if word.endswith("e") and len(word) > len(longest_word):
                longest_word = word
        return longest_word

    def archive(self):
        with zipfile.ZipFile("task2/archive.zip", "w") as z:
            z.write(self.filename)
        with zipfile.ZipFile("task2/archive.zip", "r") as z:
            return z.getinfo(self.filename)
```

Результат работы

```
---Choose task---
-1: Task1
-2: Task2
-3: Task3
-4: Task4
-5: Task5
-0: Exit
2
--Sentences count: 9
--Narrative sentences count: 5
--Interrogative sentences count: 1
--Imperative sentences count: 3
--Average word length: 4.888888888888889
--Average sentence length: 5.409090909090909
--Smileys count: 1
--All capital letters: ['T', 'I', 'A', 'H', 'L', 'I', 'T', 'S']
--Text after sequence replacement:
 This is a sample text. It contains several sentences! And some of them are questions? How interesting! Let's analyze this text.
It has narrative sentences, imperative ones, and even interrogative sentences.
The average word length is important. Smiley faces :-) are also interesting! abdsqqqdasc.
--Number of maximum length words: 1
Maximum word length: 13
--Words followed by punctuation: ['text.', 'text.', 'sentences,', 'ones,', 'sentences.', 'important.', 'abdsabcdasc.']
--Longest word that ends with "e": interrogative
```

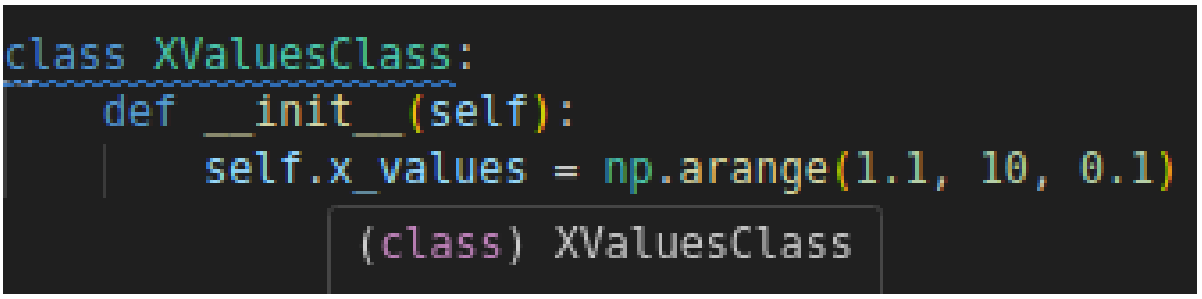
Содержимое файла



Задание 3. В соответствии с заданием своего варианта доработать программу из ЛР3, использовав 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.

Зар-т	Условие
1.	$\ln\frac{x+1}{x-1}=2\sum_{n=0}^{\infty}\frac{1}{(2n+1)x^{2n+1}}=2(\frac{1}{x}+\frac{1}{3x^3}+\frac{1}{5x^5}+...),\|x\|>1$

Класс XValuesClass



Класс Series

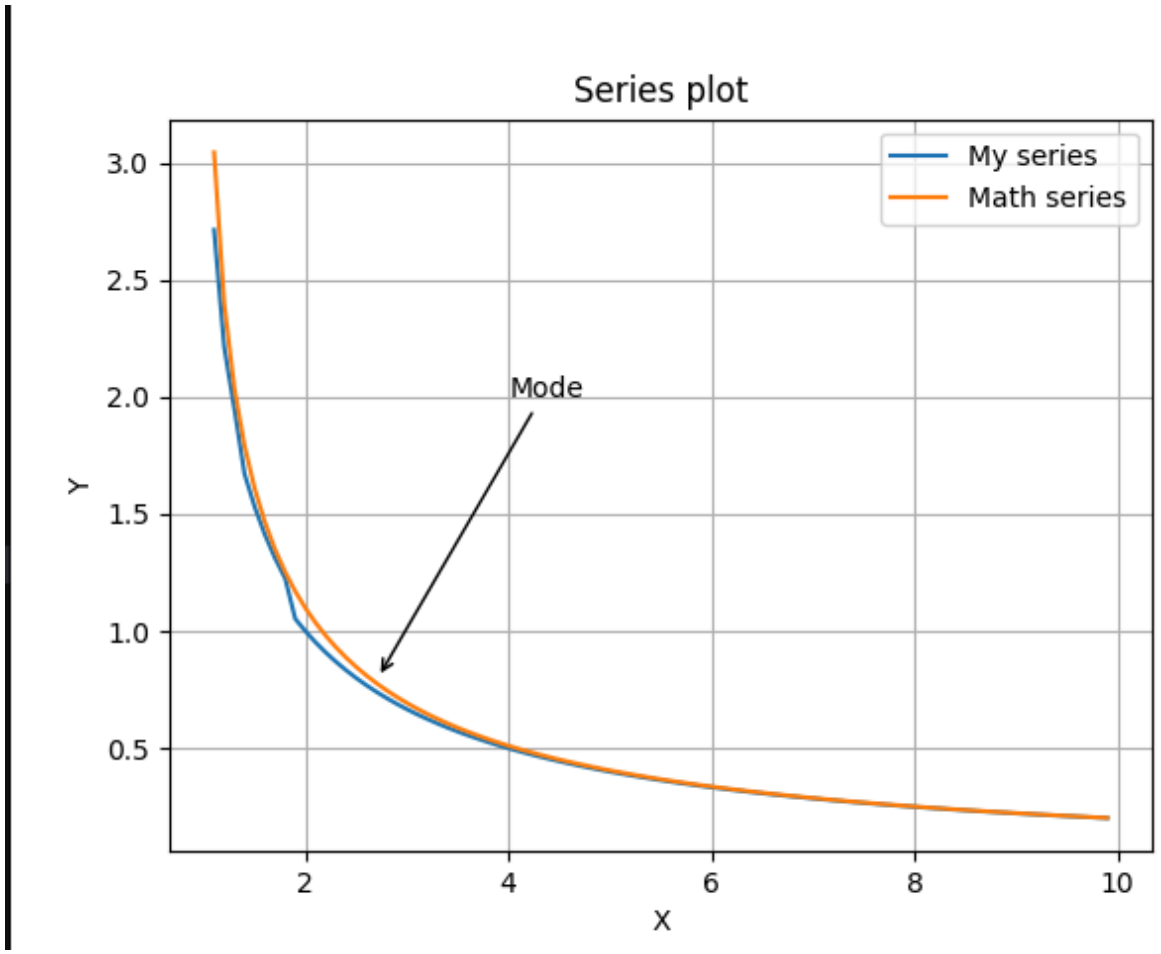
```python
class Series(XValuesClass):
    def __init__(self):
        super().__init__()
        self.eps = 1e-1

    def calculate_series(self):
        y_vals = []
        result = 0.0
        i = 0
        for x in self.x_values:
            while i <= 500:
                term = 2 / ((2 * i + 1) * (x ** (2 * i + 1)))
                if abs(term) < self.eps:
                    break
                result += term
                i += 1
            i = 0
            y_vals.append(result)
            result = 0.0
        return y_vals

    def calculate_math_series(self):
        y_vals = []
        for x in self.x_values:
            y_vals.append(math.log((x + 1)/(x - 1), math.e))
        return y_vals
```

Класс SeriesPlot

```python
class SeriesPlot(XValuesClass):
    def __init__(self, y_values, y_math_values):
        super().__init__()
        self.y_vals = y_values
        self.y_math_vals = y_math_values

    def build_plots(self):
        plt.plot(self.x_values, self.y_vals, label='My series')
        plt.plot(self.x_values, self.y_math_vals, label='Math series')

        plt.annotate('Mode', xy=(2.714043486548638, 0.8), xytext=(4, 2), arrowprops=dict(arrowstyle='->'))
        plt.legend()

        plt.savefig('task3/functions.png')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title('Series plot')
        plt.grid(True)
        plt.show()


def task3_run():
    a = Series()
    y = a.calculate_series()
    print(f"Mean: {statistics.mean(y)}\n"
          f"Median: {statistics.median(y)}\n"
          f"Dispersion: {statistics.variance(y)}\n"
          f"MSE: {statistics.stdev(y)}\n"
          f"Mode: {statistics.mode(y)}")

    y_math = a.calculate_math_series()

    plot = SeriesPlot(y, y_math)
    plot.build_plots()
```

Результаты программы

```
---Choose task---
-1: Task1
-2: Task2
-3: Task3
-4: Task4
-5: Task5
-0: Exit
3
Mean: 0.5356744721823917
Median: 0.3636363636363634
Dispersion: 0.21000380914667985
MSE: 0.4582617255964978
Mode: 2.714043486548638
```



Задание 4. В соответствии с заданием своего варианта разработать базовые классы и классы наследники.
Построить равнобедренный треугольник с основанием a и высотой h.

Класс Triangle

```
class Triangle(Figure):
    def __init__(self, height, base, color, name):
        self._h = height
        self._a = base
        self.c = Color
        self.c.color = color
        self.name = name

    def area(self):
        super().area()
        return int(1 / 2 * self._a * self._h)

    def get_info(self):
        name = self.name
        area = self.area()
        color = self.c.color
        print("Figure: {}\nColor: {}\nArea: {}".format(name, color, area))

    def plot(self):
        x = [1, 1+self._a, 1 + self._a/2,1]
        y = [1, 1, 1+self._h,1]
        fig, ax = plt.subplots()
        ax.plot(x, y)
        ax.fill(x, y, color=self.c.color, alpha=0.8)
        ax.set_aspect("equal")
        ax.set_title(self.name)
        ax.set_xlabel("X")
        ax.set_ylabel("Y")
        plt.savefig("task4/triangle.png")

        plt.show()
```

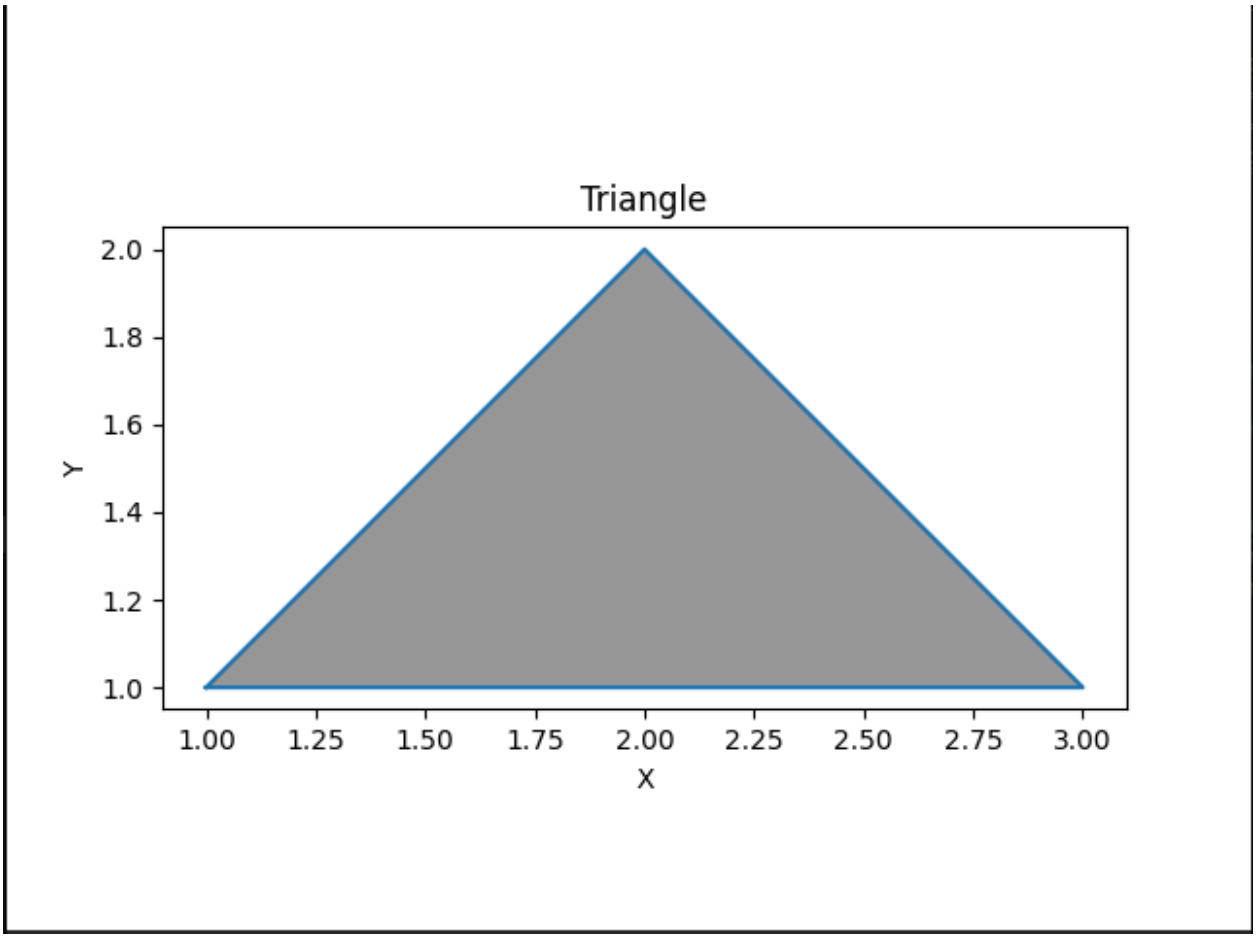
Класс Figure

```
class Figure(ABC):
    @abstractmethod
    def area(self):
        pass
```

Класс Color

```python
class Color:
    def __init__(self):
        self._color = None

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

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

    @color.deleter
    def color(self):
        del self._color
```

Результат программы



Задание 5. В соответствии с заданием своего варианта исследовать возможности библиотека NumPy при работе с массивами и математическими и статическими операциями. Сформировать целочисленную матрицу A[n,m] с помощью генератора случайных чисел (random).
Найти столбец с наименьшей суммой элементов.
Вычислить значение медианы этого столбца. Вычисление медианы выполнить двумя способами: через стандартную функцию и через программирование формулы.

Класс Matrix

```
class Matrix:
    def __init__(self):
        num_rows = np.random.randint(2, 7)
        num_cols = np.random.randint(2, 7)
        self.matrix = np.random.rand(num_rows, num_cols)

    def print_matrix(self):
        print("Matrix:")
        print(self.matrix)

    def create_zero_matrix(self, n, m):
        return np.zeros((n, m))

    def create_ones_matrix(self, n, m):
        return np.ones((n, m))

    def create_identity_matrix(self, n):
        return np.eye(n)

    def (variable) min_sum_column_index: intp
        0)
        min_sum_column_index = np.argmin(column_sums)
        return self.matrix[:, min_sum_column_index]

    def calculate_median(self, column):
        median_value = np.median(column)
        return median_value

    def calculate_median_manually(self, column):
        sorted_column = np.sort(column)
        n = len(sorted_column)
        if n % 2 == 0:
            median_value = (sorted_column[n // 2 - 1] + sorted_column[n // 2]) / 2
        else:
            median_value = sorted_column[n // 2]
        return median_value
```

```python
def calculate_mean(self, column):
    mean_value = np.mean(column)
    return mean_value

def calculate_correlation_coefficients(self):
    corr_matrix = np.corrcoef(self.matrix, rowvar=False)
    return corr_matrix

def calculate_variance(self, column):
    variance_value = np.var(column)
    return variance_value

def calculate_std_deviation(self, column):
    std_deviation = np.std(column)
    return std_deviation
```

def task5_run()

```python
def task5_run():
    matrix = Matrix()
    matrix.print_matrix()

    zero_matrix = matrix.create_zero_matrix(3, 3)
    print('Zero matrix:')
    print(zero_matrix)

    ones_matrix = matrix.create_ones_matrix(2, 2)
    print('Ones matrix:')
    print(ones_matrix)

    identity_matrix = matrix.create_identity_matrix(4)
    print('Identity matrix:')
    print(identity_matrix)

    column = matrix.get_column_with_min_sum()
    print('Columm with min sum:')
    print(column)

    median = matrix.calculate_median(column)
    print(f'Column median(standart function): {median}')

    median_manual = matrix.calculate_median_manually(column)
    print(f'Column median(formule):{median_manual}')

    mean = matrix.calculate_mean(column)
    print("Column mean:", mean)

    corr_matrix = matrix.calculate_correlation_coefficients()
    print("Matrix correlation coefficients:")
    print(corr_matrix)

    variance = matrix.calculate_variance(column)
    print(f"Column variance: {variance}")

    std_deviation = matrix.calculate_std_deviation(column)
    print("STD Deviation:", std_deviation)
```

Результат программы

```
Matrix:
[[0.46373822 0.65655651 0.24394929 0.30703959 0.31214564]
 [0.1628004  0.90437885 0.40551198 0.21320385 0.48460291]
 [0.8591778  0.29751579 0.80813765 0.82570333 0.9077722 ]
 [0.3113085  0.14851377 0.48900921 0.18639982 0.37592002]
 [0.12497631 0.53752266 0.83122412 0.1812351  0.97406112]
 [0.66479364 0.1453513  0.90336996 0.45961261 0.7534746 ]]
Zero matrix:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Ones matrix:
[[1. 1.]
 [1. 1.]]
Identity matrix:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Columm with min sum:
[0.30703959 0.21320385 0.82570333 0.18639982 0.1812351  0.45961261]
Column median(standart function): 0.26012171874671625
Column median(formule):0.26012171874671625
Column mean: 0.36219904991256574
Matrix correlation coefficients:
[[ 1.         -0.5497373   0.36602324  0.92699481  0.24877707]
 [-0.5497373   1.         -0.56477083 -0.37333959 -0.24443778]
 [ 0.36602324 -0.56477083  1.          0.4506882   0.9039672 ]
 [ 0.92699481 -0.37333959  0.4506882   1.          0.46025347]
 [ 0.24877707 -0.24443778  0.9039672   0.46025347  1.        ]]
Column variance: 0.05220351375954104
STD Deviation: 0.22848088270037176
```