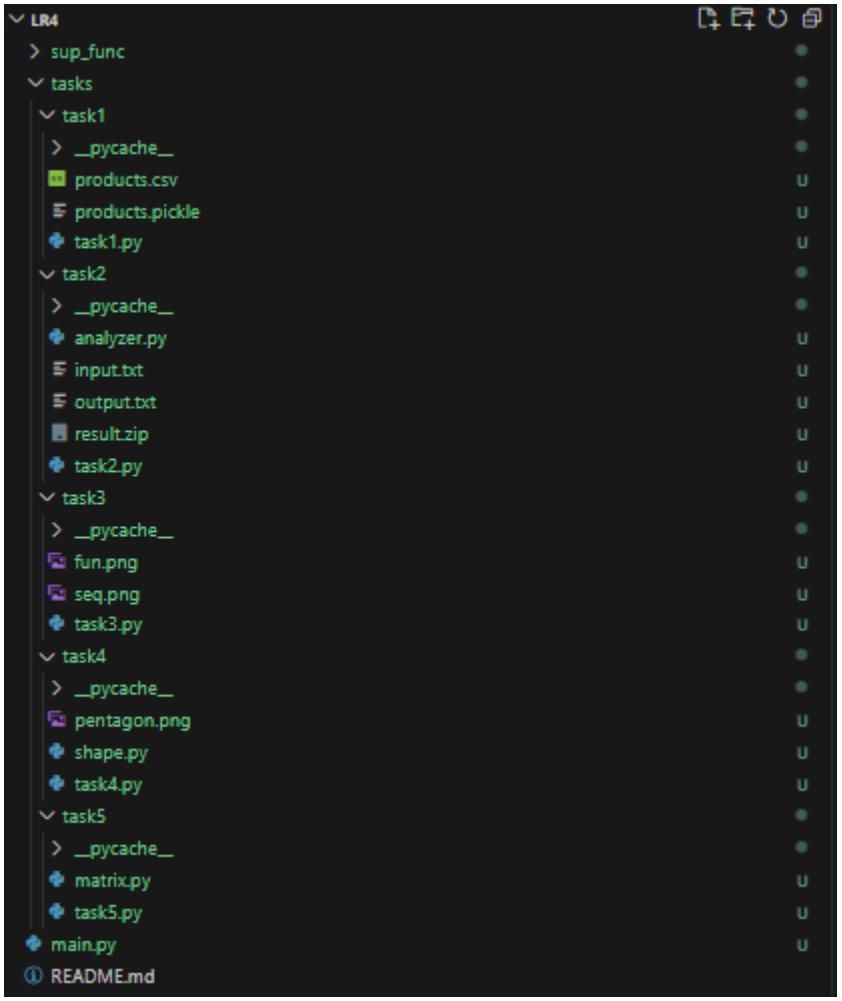
Лабораторная работа №4. Работа с файлами, классами, сериализаторами, регулярными выражениями и стандартными библиотеками.

Дриневский Кирилл гр.253502. Репозиторий: https://github.com/Drinevskiy/IGI-STRWEB.git

Структура проекта:



Вспомогательные функции: correct_input.py

sup_func > correct_input.py > input_float

```python
def input_int():
    """
    Input int and check it.

    Return:
    number(int): entered number
    """
    number = 0
    while (True):
        try:
            number = int(input())
            break
        except ValueError:
            print("Incorrect input. Try again.")
    return number


def input_float():
    """
    Input float and check it.

    Return:
    number(float): entered number
    """
    number = 0
    while (True):
        try:
            number = float(input())
            break
        except ValueError:
            print("Incorrect input. Try again.")
    return number
```

Меню: start.py

main.py > main

```python
# The program is designed to perform tasks from laboratory
# work №4 "Working with files, classes, serializers,
# regular expressions, and standard libraries.".
# Author: Drinevskiy Kirill
# Date: 14.04.2024
# Version: 1.0
from tasks.task1.task1 import task1
from tasks.task2.task2 import task2
from tasks.task3.task3 import task3
from tasks.task4.task4 import task4
from tasks.task5.task5 import task5
from sup_func.correct_input import *

def main():
    """
    The main function of the program. It provides a menu for the user to choose a task to perform.
    The tasks are performed by calling the corresponding functions.
    The menu is presented in a loop until the user chooses to exit.
    """
    while (True):
        print("First task - 1\nSecond task - 2\nThird task - 3\nFourth task - 4\nFifth task - 5\nExit - 6")
        print("Choose number to perform task", end=" ")
        choose = input_int()
        match choose:
            case 1:
                task1()
            case 2:
                task2()
            case 3:
                task3()
            case 4:
                task4()
            case 5:
                task5()
            case 6:
                break
            case _:
                print("Incorrect input. Try again.")
    print("Exit")


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

1. *Реализуйте товар (наименование товара, старая цена, новая цена). Составьте программу, определяющую, на какие товары повысятся цены и на сколько процентов. Выведите информацию о товаре, введенном с клавиатуры*

```python
import pickle
import csv

class Shop:
    """
    The Shop class represents a shop that can save and load product information.
    """
    def __init__(self, filename_pickle, filename_csv, products):
        """
        Initializes the shop with file names and products.
        """
        self.__filename_pickle = filename_pickle
        self.__filename_csv = filename_csv
        self.products = products

    @property
    def filename_txt(self):
        """
        Getter for the txt file name.
        """
        return self.__filename_pickle

    @filename_txt.setter
    def filename_txt(self, value):
        """
        Setter for the txt file name.
        """

    @pro
    def filename_csv(self):
        """
        Getter for the csv file name.
        """
        return self.__filename_csv

    @filename_csv.setter
    def filename_txt(self, value):
        """
        Setter for the csv file name.
        """
        self.__filename_csv = value

    def save_pickle(self):
        """
        Saves the product information in a pickle file.
        """
        try:
            with open(self.__filename_pickle, "wb") as fh:
                pickle.dump(self.products, fh)
        except Exception as ex:
            print("{ex}")

    def load_pickle(self):
        """
```

```
(property) filename_csv: (self: Self@Shop) -> Any
```

Getter for the csv file name.

```
    def load_pickle(self):
        """
        Loads the product information from a pickle file.
        """
        try:
            with open(self.__filename_pickle, "rb") as fh:
                return pickle.load(fh)
        except Exception as ex:
            print("{ex}")

    def save_csv(self):
        """
        Saves the product information in a csv file.
        """
        try:
            with open(self.__filename_csv, 'w', encoding="utf-8", newline="") as f:
                writer = csv.DictWriter(f, fieldnames=["Product", "Old", "New"], quoting=csv.QUOTE_ALL)
                writer.writeheader()
                for key, value in sorted(self.products.items()):
                    writer.writerow(dict(Product=key, Old=value[0], New=value[1]))
        except Exception as ex:
            print("{ex}")

    def load_csv(self):
        """
        Loads the product information from a csv file.
        """
        products = dict()
        try:
            with open(self.__filename_csv, newline="") as f:
                reader = csv.reader(f)
                for row in list(reader):
                    products[row[0]] = [row[1], row[2]]
        except Exception as ex:
            print("{ex}")
        return products

class ExtendedShop(Shop):
    """
    The ExtendedShop class extends the functionality of the Shop class, adding search and price analysis.
    """
    def __init__(self, filename_txt, filename_csv, products):
        """
        Initializes the extended shop with file names and products.
        """
        super().__init__(filename_txt, filename_csv, products)
    def search(self, query):
        """
        Searches for a product in the shop. If the product is found, it prints the old and new prices.
        If the product is not found, it prints a message.
        """
        if (query in self.products):
            print(f"{query} - old: {self.products[query][0]}, new: {self.products[query][1]}")
        else:
```

```python
    def __init__(self, analyzer):
        """
        Initializes the Archiver with an Analyzer object.
        """
        self.analyzer = analyzer

    def save_to_file(self):
        """
        Saves the analysis results to a file.
        """
        try:
            with open(Archiver.file_output, "w", encoding="utf-8") as file:
                file.write(f"Narrative: {self.analyzer.count_narrative_sentences()}\n")
                file.write(f"Interrogative: {self.analyzer.count_interrogative_sentences()}\n")
                file.write(f"Exlamation: {self.analyzer.count_exclamation_sentences()}\n")
                file.write(f"Sentences: {self.analyzer.count_sentences()}\n")
                file.write(f"Average sentence length: {self.analyzer.average_sentence_length()}\n")
                file.write(f"Average word length: {self.analyzer.average_word_length()}\n")
                file.write(f"Smiles: {self.analyzer.count_smiles()}\n")
                file.write(f"Upper case and numbers: {self.analyzer.get_upper_case_numbers()}\n")
                file.write(f"Longest word starts with 'l': {self.analyzer.get_longest_l_word()}\n")
                file.write(f"Repeatable words: {self.analyzer.repeatable_words()}\n")
        except Exception as ex:
            print("Error with writing results:", ex)

    def write_to_zip(self):
        """
        Writes the file with the analysis results to a zip archive.
        """
        try:
            with zipfile.ZipFile(Archiver.filename_zip, "w") as zip:
                zip.write(Archiver.file_output)
        except Exception as ex:
            print("Error with archives file:", ex)

    def print_info_about_zip(self):
        """
        Prints information about the file in the zip archive.
        """
        try:
            with zipfile.ZipFile(Archiver.filename_zip, "r") as zip:
                file_info = zip.getinfo(Archiver.file_output)
                print("Information about file in archive:")
                print(f"File name: {file_info.filename}")
                print(f"File size(bytes): {file_info.file_size}")
                print(f"Time of creating file: {datetime(*file_info.date_time)}")
        except Exception as ex:
            print("Error with archives file:", ex)
```

```
        if (query in self.products):
            print(f"{query} - old: {self.products[query][0]}, new: {self.products[query][1]}")
        else:
            print("Product was not found")
    def price_increase(self):
        """
        Analyzes the price increase of the products.
        Returns a dictionary with the products that had a price increase and the percentage of the increase.
        """
        products = dict()
        for key, value in self.products.items():
            if (value[0] < value[1]):
                    products[key] = round((value[1]/value[0] - 1) * 100, 2)
        return products


def task1():
    """
    The task1 function creates an ExtendedShop object, saves the product information in a csv file,
    loads the product information from the csv file, searches for a product, and prints the products that had a price increase.
    """
    products = {"apple" : [12, 15],
                "banana" : [9, 7],
                "cucumber": [7, 9],
                "pineapple": [21, 24],
                "orange": [10, 8],
                "mandarine": [8, 11]}
    filename_pickle = r"D:\Study\4 semestr\IGI\253502_Drinevskiy_8\IGI\LR4\tasks\task1\products.pickle"
    filename_csv = r"D:\Study\4 semestr\IGI\253502_Drinevskiy_8\IGI\LR4\tasks\task1\products.csv"
    shop = ExtendedShop(filename_pickle, filename_csv, products)
    shop.save_pickle()
    products2 = shop.load_pickle()
    shop.search(input("Find: "))
    print(products2)
    print(shop.price_increase())
```

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.

Вывести все слова, включающие сочетание букв верхнего регистра и цифр.

Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов, где символом может быть английская буква, цифра или знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру. 32 Примеры правильных выражений: C00l_Pass, SupperPas1. Примеры неправильных выражений: Cool_pass, C00l.

определить количество слов, состоящих из прописных букв ;

найти самое длинное слово, которое начинается на букву 'l';

вывести повторяющиеся слова

```python
import re
import zipfile
from datetime import datetime

class Analyzer:
    """
    The Analyzer class is a base class that reads a text file.
    """
    __file_input = "tasks/task2/input.txt"
    def __init__(self):
        """
        Initializes the Analyzer with the text from the file.
        """
        self._text = self.read_file()

    @property
    def text(self):
        """
        Getter for the text.
        """
        return self._text

    @text.setter
    def text(self, value):
        """
        Setter for the text.
        """
        self._text = value

    def read_file(self):
        """
        Reads the text from the file.
        """
        try:
            with open(Analyzer.__file_input, "r", encoding="utf-8") as fl:
                return fl.read()
        except Exception as ex:
            print(f"Reading error. {ex}")
            return ""

class TextAnalyzer(Analyzer):
    """
    The TextAnalyzer class extends the Analyzer class and provides methods for text analysis.
    """
    def __init__(self):
        """
        Initializes the TextAnalyzer with the text from the file.
        """
        super().__init__()

    def count_narrative_sentences(self):
        """
        Counts the number of narrative sentences in the text.
        """
        return len(re.findall(r"\.", self._text)) - 2 * (self.__count_ellipsis() + self.__count_initials())
```

```python
    def count_interrogative_sentences(self):
        """
        Counts the number of interrogative sentences in the text.
        """
        return len(re.findall(r"\?+", self._text))

    def count_exclamation_sentences(self):
        """
        Counts the number of exclamation sentences in the text.
        """
        return len(re.findall(r"\!+", self._text))

    def count_sentences(self):
        """
        Counts the total number of sentences in the text.
        """
        return self.count_narrative_sentences() + self.count_interrogative_sentences() + self.count_exclamation_sentences()

    def __count_characters(self):
        """
        Counts the number of characters in the text.
        """
        return len(re.findall(r"\w", self._text))

    def __count_words(self):
        """
        Counts the number of words in the text.
        """
        return len(re.findall(r"[A-Za-z']+", self._text))

    def average_sentence_length(self):
        """
        Calculates the average sentence length in the text.
        """
        return self.__count_characters() / self.count_sentences()

    def average_word_length(self):
        """
        Calculates the average word length in the text.
        """
        return self.__count_characters() / self.__count_words()

    def count_smiles(self):
        """
        Counts the number of smiles in the text.
        """
        return len(re.findall(r"[;:]-*(\(+|\)+|\[+|\]+)", self._text))

    def get_upper_case_numbers(self):
        """
        Finds all the upper case words and numbers in the text.
        """
        return re.findall(r"\b[A-Z1-9]+\b", self._text)
```

```
    def __count_ellipsis(self):
        """
        Counts the number of ellipsis in the text.
        """
        return len(re.findall(r"\.{3}", self._text))

    def __count_initials(self):
        """
        Counts the number of initials in the text.
        """
        return len(re.findall(r'\b[A-Z]\.\s*[A-Z]\.\s*[A-Z][A-Za-z0-9]+\b', self._text))

    def get_longest_l_word(self):
        """
        Finds the longest word that starts with 'l' in the text.
        """
        return sorted(re.findall(r"\b[l|L]+[a-zA-Z']+\b", self._text), key=len, reverse=True)[0]

    def repeatable_words(self):
        """
        Finds all the repeatable words in the text.
        """
        return set(re.findall(r"\b(\w+)\b(?=.*\b\1\b)", self._text, re.DOTALL))

    def print_info(self):
        """
        Prints the text analysis information.
        """
        print("Narrative:", self.count_narrative_sentences())
        print("Interrogative:",  self.count_interrogative_sentences())
        print("Exlamation:", self.count_exclamation_sentences())
        print("Sentences:", self.count_sentences())
        print("Average sentence length:", self.average_sentence_length())
        print("Average word length:", self.average_word_length())
        print("Smiles:", self.count_smiles())
        print("Upper case and numbers:", self.get_upper_case_numbers())
        print("Longest word starts with 'l':", self.get_longest_l_word())
        print("Repeatable words:", self.repeatable_words())

    @staticmethod
    def check_password(password):
        """
        Checks if the password is valid.
        """
        return bool(re.findall(r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d_]{8,}$", password))

class Archiver():
    """
    The Archiver class saves the analysis results to a file and a zip archive.
    """
    file_output = "tasks/task2/output.txt"
    filename_zip = "tasks/task2/result.zip"
```

```python
import tasks.task2.analyzer as Analyzer

def task2():
    """
    The task2 function creates a TextAnalyzer object, checks some passwords, prints the text analysis information,
    creates an Archiver object, saves the analysis results to a file and a zip archive, and prints information about the file in the zip archive.
    """
    analyzer = Analyzer.TextAnalyzer()
    print(Analyzer.TextAnalyzer.check_password("BHJ89_fsdjh"))
    print(Analyzer.TextAnalyzer.check_password("yt4B_89fs_djh"))
    print(Analyzer.TextAnalyzer.check_password("89fsdjh"))
    print(Analyzer.TextAnalyzer.check_password("BHJfs_djh"))
    analyzer.print_info()
    archiver = Analyzer.Archiver(analyzer)
    archiver.save_to_file()
    archiver.write_to_zip()
    archiver.print_info_about_zip()
```

3. В соответствии с заданием своего варианта(arcsin(x)) 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.

в) сохранить графики в файл

```python
import matplotlib.pyplot as plt
import numpy as np
import statistics
from sup_func.correct_input import input_float

class Function:
    """
    The Function class calculates the value of the arcsin using a series expansion with a given accuracy.
    """
    MAX_ITERATION = 500
    def __init__(self):
        """
        Initializes the Function with the input values.
        """
        self._x, self._epsilon = self.input_values()
        self._results = []
        self._n = 0
        self._values = []

    def input_values(self):
        """
        Inputs the value x and the accuracy eps.
        """
        print("Input value x:", end=" ")
        x = input_float()
        print("Input accuracy eps:", end=" ")
        eps = input_float()
        return x, eps

    def check_range(input_func):
        """
        Decorator for checking the range for arcsin.

        Parameters:
        input_func(def): function for checking the range

        Return:
        output_func(def): checked function
        """
        def output_func(*args):
            x = args[2]
            eps = args[3]
            if (abs(x) > 1):
                print("""The value of x is out of bounds.
                    x will be assigned the value of the nearest boundary.""")
                if (x > 1):
                    x = 1
                else:
                    x = -1
            return input_func(args[1], x, eps)
        return output_func


    @check_range
    def arcsin_series(self, x, eps):
```

```python
    def arcsin_series(self, x, eps):
        """
        Calculates the value of the arcsin using a series expansion with a given accuracy.

        Parameters:
        x(float): argument of function arcsin
        eps(float): accuracy
        """
        self._results = []
        self._n = 0
        self._values = []
        term = x
        result = 0
        self._values.clear()
        while abs(term) > eps and self._n < Function.MAX_ITERATION:
            self._n += 1
            result += term
            self._values.append(term)
            term *= x * x * (2 * self._n - 1) * (2 * self._n - 1) / (2 * self._n * (2 * self._n + 1))
            self._results.append(result)


class ExtendedFunction(Function):
    """
    The ExtendedFunction class extends the Function class and provides methods for drawing the function and the sequence,
    and calculating the mean, median, mode, variance, and standard deviation of the sequence.
    """
    file_fun = "tasks/task3/fun.png"
    file_seq = "tasks/task3/seq.png"


    def __init__(self):
        """
        Initializes the ExtendedFunction with the input values.
        """
        super().__init__()


    def draw_fun(self):
        """
        Draws the function arcsin(x).
        """
        x = np.arange(-1, 1, 0.01)
        y = np.arcsin(x)
        plt.plot(x, y, color='black', linewidth=2)
        plt.ylim(-1.7,1.7)
        plt.legend(['График функции'])
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('График функции arcsin(x)')
        plt.text(-1, 1,"Функция определена при |x|<=1")
        plt.grid(True)
        plt.savefig(self.file_fun)
        plt.show()
```

```python
def draw_seq(self):
    """
    Draws the sequence of the series expansion.
    """
    self.arcsin_series(self, self._x, self._epsilon)
    plt.plot([i + 1 for i in range(self._n)], self._results)
    plt.xlim(left=0, right=self._n + 1)
    plt.ylim(bottom=-1.7,top=1.7)
    plt.xlabel('Итерация')
    plt.ylabel('y')
    plt.legend(['График функции'])
    plt.title('График последовательности разложения ряда')
    plt.annotate("Достигнута заданная точность",
                 xy=(self._n, self._results[self._n - 1]),
                 xytext=(self._n / 2, 0),
                 arrowprops=dict(facecolor="blue", shrink=0.09))
    plt.grid(True)
    plt.savefig(self.file_seq)
    plt.show()


def mean_sequence(self):
    """
    Calculates the mean of the sequence.
    """
    if len(self._values) == 0:
        return None
    return statistics.mean(self._values)

def median_sequence(self):
    """
    Calculates the median of the sequence.
    """
    if len(self._values) == 0:
        return None
    return statistics.median(self._values)

def mode_sequence(self):
    """
    Calculates the mode of the sequence.
    """
    if len(self._values) == 0:
        return None
    return statistics.mode(self._values)

def variance_sequence(self):
    """
    Calculates the variance of the sequence.
    """
    if len(self._values) == 0:
        return None
    return statistics.variance(self._values)

def stdev_sequence(self):
```

```
    def stdev_sequence(self):
        """
        Calculates the standard deviation of the sequence.
        """
        if len(self._values) == 0:
            return None
        return statistics.stdev(self._values)


def task3():
    """
    Third task calculates the value of the arcsin using
    a series expansion with a given accuracy, draws the function and the sequence, and calculates
    the mean, median, mode, variance, and standard deviation of the sequence.
    """
    print("""Calculates the value of the arcsin using
a series expansion with a given accuracy.""")
    func = ExtendedFunction()
    func.draw_seq()
    func.draw_fun()
    mean = func.mean_sequence()
    median = func.median_sequence()
    mode = func.mode_sequence()
    variance = func.variance_sequence()
    stdev = func.stdev_sequence()
    print(f"Mean: {mean}\nMedian: {median}\nMode: {mode}\nVariance: {variance}\nStandard deviation: {stdev}\n")
```

4. Программа должна содержать следующие базовые функции:
1) ввод значений параметров пользователем;
2) проверка корректности вводимых данных;
3) построение, закрашивание фигуры в выбранный цвет, введенный с клавиатуры, и подпись фигуры текстом, введенным с клавиатуры;
4) вывод фигуры на экран и в файл.

Построить правильный пятиугольник со стороной a.

```python
from abc import ABC, abstractmethod
from math import pi, tan, sin, cos
import matplotlib.pyplot as plt

class Shape(ABC):
    """
    The Shape class is an abstract base class for shapes.
    """
    @abstractmethod
    def area(self):
        """
        Abstract method for calculating the area of a shape.
        """
        pass


class Color:
    """
    The Color class represents a color.
    """
    def __init__(self, color):
        """
        Initializes the Color with a color.
        """
        self._color = color

    @property
    def color(self):
        """
        Getter for the color.
        """
        return self._color

    @color.setter
    def color(self, clr):
        """
        Setter for the color.
        """
        self._color = clr

class Pentagon(Shape):
    """
    The Pentagon class represents a pentagon and extends the Shape class.
    """
    def __init__(self, name, a, color):
        """
        Initializes the Pentagon with a name, a side length, and a color.
        """
        self._name = name
        self._a = a
        self._color = Color(color)

    def __format__(self, format_spec):
        """
        Formats the Pentagon information.
```

```
    def __format__(self, format_spec):

        if format_spec == 'a':
            return str(self._a)
        elif format_spec == 'color':
            return self._color.color
        elif format_spec == 'name':
            return self._name
        elif format_spec == 'area':
            return str(self.area())
        return self._name + ', ' + str(self._a) + ', ' + self._color.color + ', ' + str(self.area())

    @property
    def name(self):
        """
        Getter for the name.
        """
        return self._name

    @name.setter
    def name(self, value):
        """
        Setter for the name.
        """
        self._name = value

    @property
    def a(self):
        """
        Getter for the side length.
        """
        return self._a

    @a.setter
    def a(self, value):
        """
        Setter for the side length.
        """
        if not isinstance(value, int) or value <= 0:
            value = 1
        self._a = value

    def area(self):
        """
        Calculates the area of the pentagon.
        """
        return (5.0/4)*self._a**2/tan(pi/5)

    def get_pentagon_coordinates(self):
        """
        Calculates the coordinates of the pentagon.
        """
        x = 0
        y = 0
```

```
    y = 0
    r = self._a / (2 * sin(pi / 5))
    return [x + r * cos(2 * pi * i / 5) for i in range(1, 6)], [y + r * sin(2 * pi * i / 5) for i in range(1, 6)]


def draw_shape(self, text):
    """
    Draws the pentagon and saves it to a file.
    """
    filename = "tasks/task4/pentagon.png"
    fig, ax = plt.subplots()

    x_coords, y_coords = self.get_pentagon_coordinates()

    try:
        ax.fill(x_coords, y_coords, color=self._color.color)
    except Exception as e:
        print("An error occurred while filling the shape:", str(e))
        return

    ax.text(min(x_coords), min(y_coords), text, fontsize=9)
    plt.savefig(filename)
    plt.show()
```

```
import sup_func.correct_input
from tasks.task4.shape import Pentagon
import sup_func

def task4():
    """
    The task4 function performs the fourth task. It inputs the figure name, side length, and color,
    creates a Pentagon object, prints the Pentagon information, inputs the text on the graphic, and draws the Pentagon.
    """
    name = input("Input figure name: ")
    print("Input side length:", end=" ")
    a = sup_func.correct_input.input_int()
    color = input("Input figure color: ")
    pentagon = Pentagon(name, a, color)
    print("Name: {:name} Side: {:a} Color: {:color} Area: {:area}".format(pentagon, pentagon, pentagon, pentagon))
    text = input("Input text on graphic: ")
    pentagon.draw_shape(text)
```

5. Отсортировать матрицу по убыванию элементов последнего столбца. Вычислить среднее значение элементов последнего столбца. Ответ округлите до сотых. Вычисление среднего значения выполнить двумя способами: через стандартную функцию и через программирование формулы

```python
import numpy as np

class Matrix:
    """
    The Matrix class represents a matrix with n rows and m columns.
    """
    def __init__(self, n, m):
        """
        Initializes the Matrix with n rows and m columns, and fills it with random integers between -20 and 20.
        """
        self._n = n
        self._m = m
        self._matrix = np.random.randint(-20, 20, (n, m))

    def __str__(self):
        """
        Returns a string representation of the Matrix.
        """
        return f"Количество строк: {self._n}, количество столбцов: {self._m}.\n{self._matrix}"

    def sort_by_last_column(self):
        """
        Sorts the rows of the Matrix in descending order by the last column.
        """
        sorted_indices = np.argsort(self._matrix[:, -1])[::-1]
        self._matrix = self._matrix[sorted_indices]

class StatisticMatrixMixin:
    """
    The StatisticMatrixMixin class is a mixin that provides statistical methods for a matrix.
    """
    def mean(self):
        """
        Calculates the mean of the matrix.
        """
        return np.mean(self._matrix)

    def mean_last_column(self):
        """
        Calculates the mean of the last column of the matrix.
        """
        return np.mean(self._matrix[:, -1])

    def my_mean_last_column(self):
        """
        Calculates the mean of the last column of the matrix using the number of rows.
        """
        return np.sum(self._matrix[:, -1]) / self._n

    def median(self):
        """
        Calculates the median of the matrix.
        """
        return np.median(self._matrix)
```

```
    def corrcoef(self):
        """
        Calculates the correlation coefficient of the matrix.
        """
        return np.corrcoef(self._matrix)

    def var(self):
        """
        Calculates the variance of the matrix.
        """
        return np.var(self._matrix)

    def std(self):
        """
        Calculates the standard deviation of the matrix.
        """
        return np.std(self._matrix)

class MyMatrix(StatisticMatrixMixin, Matrix):
    """
    The MyMatrix class extends the Matrix class and includes the StatisticMatrixMixin.
    """
    pass
```

```
import tasks.task5.matrix as mtr
import sup_func.correct_input

def task5():
    """
    The task5 function performs the fifth task. It inputs the row size and column size, creates a MyMatrix object,
    prints the matrix, sorts the matrix by the last column, prints the sorted matrix, calculates the mean of the last column
    using numpy and a custom function, and prints the calculated means.
    """
    print("Input row size:", end=" ")
    n = sup_func.correct_input.input_int()
    print("Input column size:", end=" ")
    m = sup_func.correct_input.input_int()
    matrix = mtr.MyMatrix(n, m)
    print(matrix)
    matrix.sort_by_last_column()
    print(matrix)
    mean = round(matrix.mean_last_column(), 2)
    my_mean = round(matrix.my_mean_last_column(), 2)
    print(f"Mean(numpy) = {mean}")
    print(f"Mean(my function) = {my_mean}")
```