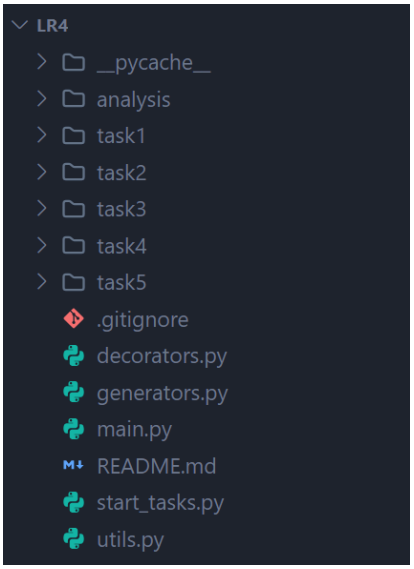
Лабораторная работа №4

Тема: работа с файлами, классами, сериализаторами, регулярными выражениями и стандартными библиотеками.
Цель: освоить базовый синтаксис языка Python, приобрести навыки работы с файлами, классами, сериализаторами, регулярными выражениями и стандартными библиотеками и закрепить их на примере разработки интерактивных приложений.
Выполнил: студент группы 253502 Красёв П.А.
Вариант 11.

Структура лабораторной работы:



Файл main.py:

```python
from start_tasks import TasksStarter

# Lab4. Working with files, classes, serializers, regular expressions and standard libraries.
# Performed by a student of group 253502, Krasyov Pavel.
# Fulfillment date: 25.03.2024


def main():
    starter = TasksStarter()
    while True:
        try:
            task = int(input(
                'Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: '))
            if task < 0 or task > 6:
                raise Exception
        except:
            print(
                'Incorrect input, try again! Input only integer numbers between 0 and 6!\n')
            continue
        if task == 1:
            starter.runTask1()
            continue
        elif task == 2:
            starter.runTask2()
            continue
        elif task == 3:
            starter.runTask3()
            continue
        elif task == 4:
            starter.runTask4()
            continue
        elif task == 5:
            starter.runTask5()
            continue
        elif task == 6:
            starter.runAnalisys()
            continue
        elif task == 0:
            break
    print('Program finished!')


if __name__ == '__main__':
    main()

```

Функция-генератор матрицы для задания 5:

```python
import numpy as np


def matrixGenerator(rows: int, columns: int):
    return np.random.randint(-150, 150, size=(rows, columns))

```

Декоратор для функций запуска заданий:

```python
from utils import stopFunction


def RunTask(Class):
    def RunningDecorator(func):
        def RunLoop(self):
            print(func.__doc__)
            obj = Class()
            while True:
                res = func(self, obj)
                if res is not None or stopFunction('finish task') != "0":
                    break
            print('Task exit')
        return RunLoop
    return RunningDecorator

```

Функции проверки ввода:

```python
from matplotlib import pyplot as plt


def stopFunction(action: str = 'continue') -> str:
    ''' Function stops running task. '''
    return input(f'Do you want to {action}? (0 -> No | anything else -> Yes): ')


def validateNumberInput(inputMessage: str, errorMessage: str, numberType, condition=lambda x: type(x) == str) -> int:
    ''' Function validates input of a number with a given type and condition. '''
    while True:
        try:
            N = numberType(
                input(f'Enter {inputMessage}: '))
            if condition(N):
                print(f'Wrong input -> ({errorMessage})! Try again.\n')
                continue
        except ValueError:
            print('Wrong input! Try again.\n')
            continue
        break
    return N


def validateColorInput(figure: str) -> str:
    ''' Function validates input of figure color. '''
    while True:
        color = input(f'Input {figure} color: ')
        try:
            plt.plot([], [], color=color)
        except ValueError:
            print('Color is not available!\n')
            continue
        break
    return color
```

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;
Вам дан словарь, состоящий из слов, расположенных парами. Каждое слово является синонимом к парному ему слову. Все слова в словаре различны. Для последнего слова из словаря определите его синоним. Выведите информацию о слове, введенном с клавиатуры.

Функция запускающая задание:

```python
@RunTask(Task1)
def runTask1(self, task: Task1) -> str | None:
    ''' Task1\nA dictionary consisting of words arranged in pairs is given. Each word is a synonym for its paired
    word. All words in the dictionary are different. For the last word in the dictionary, determine its synonym.
    Display information about the word entered from the keyboard. '''
    var = validateNumberInput('type (1 -> csv file | 2 -> pickle module | 0 -> Exit function)',
                              '1 -> csv file | 2 -> pickle module | 0 -> Exit function', int,
                              condition=lambda x: x != 0 and x != 1 and x != 2)
    if var == 1:
        dictionary = CsvDictionary(f'{task.path}csv')
    elif var == 2:
        dictionary = PickleDictionary(f'{task.path}pkl')
    else:
        return 'stop'
    task(dictionary)
```

Исходный словарь и родительский класс словарей:

```python
synonyms_dict = dict([["быстрый", "скорый"],
                      ["медленный", "неспешный"],
                      ["умный", "интеллектуальный"],
                      ["глупый", "неразумный"],
                      ["смелый", "отважный"],
                      ["трусливый", "боязливый"]])


class BaseDictionary():
    def __init__(self, FILENAME: str) -> None:
        ''' Inits class object. '''
        self.filename = FILENAME

    def getDictionary(self) -> dict[str, str]:
        ''' Method returns dictionary from the file. '''
        pass

    def sortDictionary(self) -> None:
        ''' Method sorts dictionary and rewrites it in the file. '''
        pass

    def getLastSynonym(self) -> str:
        ''' Method returns last word-synonym from the dictionary. '''
        return list(self.getDictionary().values())[-1]

    def getWordInfo(self, word: str) -> None:
        ''' Method returns info about the word according on the file. '''
        dictionary = self.getDictionary()
        values: list[str] = list(dictionary.values())
        if word in values:
            try:
                print(
                    f'Pair: {list(dictionary.keys())[values.index(word)]} - {word}')
            except ValueError:
                print('Word isn\'t found!')
            finally:
                return
        synonym: str | None = dictionary.get(word)
        print(
            f'Pair: {word} - {synonym}' if synonym is not None else 'Word isn\'t found!')

```

Класс csv словаря:

```python
import csv
from task1.baseDictionary import BaseDictionary, synonyms_dict


class CsvDictionary(BaseDictionary):
    def __init__(self, FILENAME: str) -> None:
        ''' Inits class object. '''
        super().__init__(FILENAME)
        with open(FILENAME, "w", newline='') as file:
            csv.writer(file).writerows(synonyms_dict.items())

    def getDictionary(self) -> dict[str, str]:
        ''' Method returns dictionary from csv file. '''
        with open(self.filename, 'r', newline='\n') as file:
            reader = csv.reader(file)
            synonyms: dict[str, str] = dict()
            for row in reader:
                synonyms[row[0]] = row[1]
            return synonyms

    def sortDictionary(self) -> None:
        ''' Sorts dictionary inside the csv file. '''
        file_dict = self.getDictionary()
        with open(self.filename, 'w', newline='') as file:
            csv.writer(file).writerows(
                {k: file_dict[k] for k in sorted(file_dict)}.items())

```

Класс pickle словаря:

```python
import pickle
from task1.baseDictionary import BaseDictionary, synonyms_dict


class PickleDictionary(BaseDictionary):
    def __init__(self, FILENAME: str) -> None:
        ''' Inits class object. '''
        super().__init__(FILENAME)
        with open(FILENAME, 'wb') as file:
            pickle.dump(synonyms_dict, file)

    def getDictionary(self) -> dict[str, str]:
        ''' Method returns dictionary from pkl file. '''
        with open(self.filename, 'rb') as file:
            return pickle.load(file)

    def sortDictionary(self) -> None:
        ''' Sorts dictionary inside the csv file. '''
        file_dict = self.getDictionary()
        with open(self.filename, 'wb') as file:
            pickle.dump(
                {k: file_dict[k] for k in sorted(file_dict)}, file)

```

Общий класс задания 1:

```python
from task1.csvDictionary import CsvDictionary
from task1.pickleDictionary import PickleDictionary
from utils import stopFunction


class Task1():
    def __init__(self) -> None:
        ''' Inits object of the class. '''
        self.path = './task1/file.'

    def __call__(self, dictionary: CsvDictionary | PickleDictionary) -> None:
        ''' Runs class methods. '''
        self.printDictionary(dictionary)
        self.getLastWord(dictionary)
        dictionary.sortDictionary()
        self.printDictionary(dictionary, 'Sorted ')
        self.getLastWord(dictionary)
        self.getWordInfo(dictionary)

    def printDictionary(self, dictionary: CsvDictionary | PickleDictionary, msg: str = '') -> None:
        ''' Method returns loaded from file dictionary. '''
        print(f'{msg}Dictionary:')
        for words in dictionary.getDictionary().items():
            print(f'{words[0]} - {words[1]}')
        print()

    def getLastWord(self, dictionary: CsvDictionary | PickleDictionary) -> None:
        ''' Method returns last word-synonym from dictionary. '''
        print('Last word-synonym:')
        print(f'{dictionary.getLastSynonym()}')
        print()

    def getWordInfo(self, dictionary: CsvDictionary | PickleDictionary) -> None:
        ''' Method returns word\'s synonym from the dictionary if it\'s possible. '''
        while True:
            word = input('Enter word to find: ')
            dictionary.getWordInfo(word)
            print()
            if stopFunction() == '0':
                break

```

Пример использования:

```
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 1
 Task1
A dictionary consisting of words arranged in pairs is given. Each word is a synonym for its paired word. All words in the dictionary are different. For th
e last word in the dictionary, determine its synonym. Display information about the word entered from the keyboard.
Enter type (1 -> csv file | 2 -> pickle module | 0 -> Exit function): 1
Dictionary:
быстрый - скорый
глупый - неразумный
медленный - неспешный
смелый - отважный
трусливый - боязливый
умный - интеллектуальный

Last word-synonym:
интеллектуальный

Enter word to find: глупый
Pair: глупый - неразумный

Do you want to continue? (0 -> No | anything else -> Yes): 0
Do you want to finish task? (0 -> No | anything else -> Yes): 0
Enter type (1 -> csv file | 2 -> pickle module | 0 -> Exit function): 2
Dictionary:
быстрый - скорый
глупый - неразумный
медленный - неспешный
смелый - отважный
трусливый - боязливый
умный - интеллектуальный

Last word-synonym:
интеллектуальный

Enter word to find: боязливый
Pair: трусливый - боязливый

Do you want to continue? (0 -> No | anything else -> Yes): 0
Do you want to finish task? (0 -> No | anything else -> Yes): yes
Task exit
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 0
Program finished!
PS D:\IGI\253502_KRASYOV_11\IGI\LR4>
```

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 менее одной) одинаковых скобок из следующего набора: «(», «)», «[», «]»;
    - внутри смайлика не может встречаться никаких других символов.

Найти в тексте:

- вывести все слова, включающие символы, лежащих в диапазоне от 'a' до 'o' и цифры;

- проверить, является ли заданная строка шестизначным числом, записанным в десятичной системе счисления без нулей в старших разрядах. Примеры правильных выражений: 123456, 234567. Примеры неправильных выражений: 1234567, 12345;
- определить количество слов, заключенных в кавычки;
- определить, сколько раз повторяется каждая буква;
- вывести в алфавитном порядке все словосочетания, отделенные запятыми.

Функция, запускающая задание:

```python
@RunTask(Task2)
def runTask2(self, task: Task2) -> None:
    ''' Task2\nRead text from the source file. Using regular expressions, obtain the required information display
    it on the screen and save it to another file. Archive the result file using the zipfile module and ensure that
    information about the file is retrieved in the archive. '''
    task()
```

Абстрактный класс анализатора текста:

```python
from abc import ABC


class Analiser(ABC):
    def countSentences(self) -> int:
        ''' Method returns amount of sentences. '''
        pass

    def countTypesSentences(self) -> tuple[int, int, int]:
        ''' Method returns amount of each type sentence. '''
        pass

    def avgSentenceLen(self) -> float:
        ''' Method returns average sentence length. '''
        pass

    def avgWordLen(self) -> float:
        ''' Method return average word length. '''
        pass

    def countSmiles(self) -> int:
        ''' Method returns ammount of smiles in the text. '''
        pass

    def getMatchWords(self) -> list[str]:
        ''' Method returns words with letters from \'a\' to \'o\' or digits. '''
        pass

    def checkStr(self, input_string: str) -> bool:
        ''' Method checks if input string is 6-digits number in decimal system without leading zeros. '''
        pass

    def countQuotesWords(self) -> int:
        ''' Method returns amount of words in quotes. '''
        pass

    def countLetters(self) -> dict[str, int]:
        ''' Method returns amount of each letter in text. '''
        pass

    def getPhrases(self) -> list[str]:
        ''' Method returns phrases in commas from the text. '''
        pass
```

Класс анализатора текста, реализующий методы абстрактного класса:

```python
import re
from collections import Counter
from task2.baseAnaliser import Analiser


class TextAnaliser(Analiser):
    def __init__(self, text: str):
        ''' Inits class object. '''
        self.text = text
        self.sentences = re.split(r'(?<=[.!?])\s', text)
        self.words = re.findall(r'\w+', self.text)

    def countSentences(self) -> int:
        ''' Method returns amount of sentences. '''
        return len(self.sentences)

    def countTypesSentences(self) -> tuple[int, int, int]:
        ''' Method returns amount of each type sentence. '''
        return (len([1 for sent in self.sentences if sent.endswith('.')]),
                len([1 for sent in self.sentences if sent.endswith('?')]),
                len([1 for sent in self.sentences if sent.endswith('!')]))

    def avgSentenceLen(self) -> float:
        ''' Method returns average sentence length. '''
        sentence_lengths = [len(word) for word in self.words]
        return round(sum(sentence_lengths) / len(self.sentences), 3) if self.sentences else 0

    def avgWordLen(self) -> float:
        ''' Method return average word length. '''
        words_lengths = [len(word) for word in self.words]
        return round(sum(words_lengths) / len(self.words), 3) if self.words else 0

    def countSmiles(self) -> int:
        ''' Method returns ammount of smiles in the text. '''
        return len(re.findall(r'[:;]-*[()\[\]]+', self.text))

    def getMatchWords(self) -> list[str]:
        ''' Method returns words with letters from \'a\' to \'o\' or digits. '''
        return re.findall(r'\b\w*([a-o]+\w*\d+|\d+\w*[a-o]+)\w*\b', self.text)

    def checkStr(self, input_string: str) -> bool:
        ''' Method checks if input string is 6-digits number in decimal system without leading zeros. '''
        return bool(re.match(r'^[1-9]\d{5}$', input_string))

    def countQuotesWords(self) -> int:
        ''' Method returns amount of words in quotes. '''
        return len(re.findall(r'\s(\"|\'|«)(.+?)(\"|\'|\»)', self.text))

    def countLetters(self) -> dict[str, int]:
        ''' Method returns amount of each letter in text. '''
        return Counter(re.findall(r'[a-z]', self.text.lower()))

    def getPhrases(self) -> list[str]:
        ''' Method returns phrases in commas from the text. '''
        return sorted([ph[0][2:-1].lower() for ph in re.findall(r'(\,(\s+\w+){2,}\,)', self.text)])
```

Общий класс задания 2:

```python
from task2.textAnaliser import TextAnaliser
from task2.baseAnaliser import Analiser
from utils import validateNumberInput
from zipfile import ZIP_DEFLATED, ZipFile


class Task2(Analiser):
    _border: str = ('*' * 80) + "\n"

    def __init__(self) -> None:
        ''' Inits class object. '''
        self.path = './task2/'
        self.results: list[str] = list()
        with open(f'{self.path}file.txt', 'r', encoding='utf-8') as file:
            text = file.read()
        self.analiser = TextAnaliser(text)

    def __call__(self) -> None:
        ''' Runs class methods. '''
        self.results.clear()
        self.countSentences()
        self.countTypesSentences()
        self.avgSentenceLen()
        self.avgWordLen()
        self.countSmiles()
        self.getMatchWords()
        self.checkStr()
        self.countQuotesWords()
        self.countLetters()
        self.countPhrases()
        self.saveResults()
        self.zipFile()

    def countSentences(self) -> None:
        ''' Method returns ammount of sentences in the text. '''
        self.results.append(
            f'Sentenses ammount: {self.analiser.countSentences()}\n' + self._border)

    def countTypesSentences(self) -> None:
        ''' Method returns ammount of each type sentence. '''
        ammount = self.analiser.countTypesSentences()
        self.results.append(f'''Declarative sentenses ammount: {ammount[0]}
Interrogative sentences ammount: {ammount[1]}
Incentive sentenses ammount: {ammount[2]}\n''' + self._border)

    def avgSentenceLen(self) -> None:
        ''' Method returns average sentence length. '''
        self.results.append(
            f'Average sentence length: {self.analiser.avgSentenceLen()}\n' + self._border)
```

```python
    def avgWordLen(self) -> None:
        ''' Method return average word length. '''
        self.results.append(
            f'Average word length: {self.analiser.avgWordLen()}\n' + self._border)

    def countSmiles(self) -> None:
        ''' Method returns ammount of smiles in the text. '''
        self.results.append(
            f'Ammount of smiles: {self.analiser.countSmiles()}\n' + self._border)

    def getMatchWords(self) -> None:
        ''' Method returns words with letters from \'a\' to \'o\' or digits. '''
        result_str = 'Words with letters from \'a\' to \'o\' and digits:\n'
        for word in self.analiser.getMatchWords():
            result_str += word + "\n"
        self.results.append(result_str + self._border)

    def checkStr(self) -> None:
        ''' Method checks if input string is 6-digits number in decimal system without leading zeros. '''
        input_string = str(validateNumberInput(
            'integer number string', '', int))
        res = '' if self.analiser.checkStr(input_string) else ' not'
        self.results.append(
            f'String {input_string} is{res} valid\n' + self._border)
        print()

    def countQuotesWords(self) -> None:
        ''' Method returns amount of words in quotes. '''
        self.results.append(
            f'Ammount of words in quotes: {self.analiser.countQuotesWords()}\n' + self._border)

    def countLetters(self) -> None:
        ''' Method returns amount of each letter in text. '''
        result_str = 'Ammount of each letter:\n'
        for res in self.analiser.countLetters().items():
            result_str += f'{res[0]} - {res[1]}\n'
        self.results.append(result_str + self._border)

    def countPhrases(self) -> None:
        ''' Method returns phrases in commas from the text. '''
        result_str = 'Phrases in commas:\n'
        for phrase in self.analiser.getPhrases():
            result_str += phrase + "\n"
        self.results.append(result_str + self._border)
```

```python
    def saveResults(self) -> None:
        ''' Method saves results to a file. '''
        print('Analised text:')
        print(self.analiser.text)
        print("\nResults:")
        with open(f'{self.path}results.txt', 'w', newline='\n', encoding='utf-8') as file:
            for res in self.results:
                file.write(res)
                print(res)

    def zipFile(self) -> None:
        ''' Method zips results. '''
        with ZipFile(f'{self.path}results.zip', 'w', compression=ZIP_DEFLATED) as file:
            file.write(f'{self.path}results.txt')
            print('Zip file info:')
            for item in file.infolist():
                print(f'Filename: {item.filename}')
                print(f'''Last change: {datetime(item.date_time[0], item.date_time[1], item.date_time[2],
                                                item.date_time[3], item.date_time[4], item.date_time[5])}''')
                print(f'Compress type: {item.compress_type}')
                print(f'Compress size: {item.compress_size}')
                print(f'File size: {item.file_size}')
```

Пример использования:

Исходный текст:

```
When I have free time, I often listen to my favourite composer, Wolfgang Amadeus Mozart, His works fill me with new energy, help me to relax and raise my mood. He was a very talented musician. His father taught him the violin ;--), piano and musical theory. Little Mozart began to write music at the age of four. He wrote his first opera when he was eleven. When Mozart became an adult, he moved to Vienna. He had been successful in this town as a :))) child prodigy, but as an adult he found it difficult to find a job.
In Vienna he met Haydn, who became his second father. Haydn supported the young composer and h3lped him in his musical career. Mozart's operas became very popular in the city. He did not spend much time thinking about his next composition. Musical ideas sprang from his mind and he just had ;-[[[[ to write them down. At this time he married Constance Weber and wrote one of his most fam0us works — «C minor» composition qrst123.
Mozart enjoyed a successful career. He worked a lot. He visited Prague with his operas. Writing his last work 'Requiem', commissioned by an unusual stranger, that it was his own requiem. He did not manage to finish his work and died at the age of 35 from poor health. The "Requiem" was completed by one of his pupils, Sussmary. Mozart's 49 symphonies and 18 operas are now world famous and are considered to have healing power.

```

Результаты анализа:

```
Sentenses ammount: 21
********************************************************************************
Declarative sentenses ammount: 20
Interrogative sentences ammount: 0
Incentive sentenses ammount: 0
********************************************************************************
Average sentence length: 51.095
********************************************************************************
Average word length: 4.292
********************************************************************************
Ammount of smiles: 3
********************************************************************************
Words with letters from 'a' to 'o' and digits:
3lped
m0
********************************************************************************
String 12345 is not valid
********************************************************************************
Ammount of words in quotes: 3
********************************************************************************
Ammount of each letter:
w - 29
h - 64
e - 121
n - 69
i - 83
a - 88
v - 11
f - 24
r - 62
t - 78
m - 50
o - 83
l - 30
s - 72
y - 16
u - 35
c - 29
p - 25
g - 16
d - 45
z - 6
k - 6
x - 2
b - 11
j - 3
q - 4
********************************************************************************
```

```
Phrases in commas:
commissioned by an unusual stranger
his works fill me with new energy
i often listen to my favourite composer
**************************************************************************

Zip file info:
Filename: task2/results.txt
Last change: 2024-04-11 20:02:22
Compress type: 8
Compress size: 390
File size: 1455
Do you want to finish task? (0 -> No | anything else -> Yes): yes
Task exit
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 0
Program finished!
PS D:\IGI\253502_KRASYOV_11\IGI\LR4>
```

Задание 3. В соответствии с заданием своего варианта составить программу для вычисления значения функции с помощью разложения функции в степенной ряд. Задать точность вычислений eps. Предусмотреть максимальное количество итераций, равное 500.

$$ln((x + 1)/(x - 1)) = 2 * \sum_{n=0}^{inf} 1/((2n + 1) * x^{2n+1}),\ |x|>1$$

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

Функция, запускающая задание:

```
32    @RunTask(Task3)
33    def runTask3(self, task: Task3) -> str | None:
34        ''' Task3\nWith a given accuracy calculate the value of a function using Tailor's row. Determine additional
          parameters: arithmetic mean, median, mode, dispersion, standard deviation of the sequence. Using the
          `matplotlib` library, draw charts of different colors on the same coordinate axis'''
35        return task()
```

Класс, вычисляющий дополнительный параметры последовательности:

```python
from statistics import fmean, median, mode, variance, stdev


class RowAnaliser():
    def __init__(self) -> None:
        ''' Inits class object. '''
        self.row: list[float] = list()

    def __call__(self) -> tuple[float, float, float, float, float] | str:
        ''' Runs class methods. '''
        if len(self.row) == 0:
            return 'Tailor\'s row is empty!'
        return (self.mean(), self.median(), self.mode(), self.variance(), self.stdev())

    def mean(self) -> float | str:
        ''' Method counts arithmetic mean of sequence. '''
        return round(fmean(self.row), 4) if len(self.row) > 0 else 'Tailor\'s row is empty!'

    def median(self) -> float | str:
        ''' Method returns sequence median. '''
        return round(median(self.row), 4) if len(self.row) > 0 else 'Tailor\'s row is empty!'

    def mode(self) -> float | str:
        ''' Method returns sequence mode. '''
        return round(mode(self.row), 4) if len(self.row) > 0 else 'Tailor\'s row is empty!'

    def variance(self) -> float | str:
        ''' Method counts sequence variance. '''
        if len(self.row) == 0:
            return 'Tailor\'s row is empty!'
        elif len(self.row) < 2:
            return 'Can\'t find variance. 2 and more elements needed.'
        return round(variance(self.row), 4)

    def stdev(self) -> float | str:
        ''' Method counts standart deviance of sequence. '''
        if len(self.row) == 0:
            return 'Tailor\'s row is empty!'
        elif len(self.row) < 2:
            return 'Can\'t find standart deviance. 2 and more elements needed.'
        return round(stdev(self.row), 4)

```

Класс, вычисляющий значение функции и сумму ряда Тейлора в точке:

```python
from math import log
from task3.rowAnaliser import RowAnaliser


class TailorSum(RowAnaliser):
    def __init__(self) -> None:
        ''' Inits class object. '''
        super().__init__()

    def __call__(self):
        ''' Clears list with row elements. '''
        self.row.clear()

    def countMath(self, x: float):
        ''' Method returns function value in the x point. '''
        return log((x + 1)/(x - 1))

    def countTailor(self, x: float, eps: float, mathResult: float) -> tuple[float, int]:
        ''' Function that counts sum of Tailor's row with the required accuracy. '''
        tailorSum = 0
        N = -1
        while N + 1 < 500:
            N += 1
            element = 1/((2*N+1)*x**(2*N+1))
            tailorSum += element
            if (abs(2*tailorSum - mathResult) >= eps/10):
                continue
            break
        self.row.append(2*tailorSum)
        return 2*tailorSum, N+1

    def getAdditionalParams(self) -> str:
        ''' Function returns additional parameters of sequence. '''
        params = super().__call__()
        if type(params) == str:
            return params
        return f'Additional parameters:\nArithmetics mean: {params[0]}\n\
Median: {params[1]}\nMode: {params[2]}\nVariance: {params[3]}\nStandard deviation: {params[4]}'

```

Общий класс задания 3:

```python
from utils import validateNumberInput
import matplotlib.pyplot as plt
from statistics import median
from task3.tailorSum import TailorSum


class Task3():
    def __init__(self) -> None:
        ''' Inits class object. '''
        self.tailor = TailorSum()
        self.mathResults = list()
        self.tailorResults = list()
        self.iterations = list()

    def __call__(self) -> None | str:
        ''' Runs class methods. '''
        self.validateInput()
        if self.eps == 0:
            return 'stop'
        self.tailor()
        self.displayConsole()
        self.displayCharts()

    def validateInput(self) -> None:
        ''' Method for validation input of accuracy and x value. '''
        self.eps = validateNumberInput(
            'accuracy (0 < acc < 1) (0 -> Exit function)', '0 < acc < 1', float, condition=lambda x: x < 0 or x >= 1)
        if self.eps == 0:
            return
        start = validateNumberInput(
            'start x (|x| > 1)', '|x| must be bigger than 1', float, condition=lambda x: abs(x) <= 1)
        while True:
            end = validateNumberInput(
                'end x (|x| > 1 & x > start)', '|x| must be bigger than 1', float, condition=lambda x: abs(x) <= 1)
            if end <= start:
                print('Wrong input -> (end point > start point)! Try again!\n')
                continue
            break
        N = validateNumberInput(
            'number of points (N > 0)', 'N - integer positive number', int, condition=lambda x: x <= 0)
        h = (end - start) / N
        self.x = [round(start + i*h, 4) for i in range(N + 1)]
```

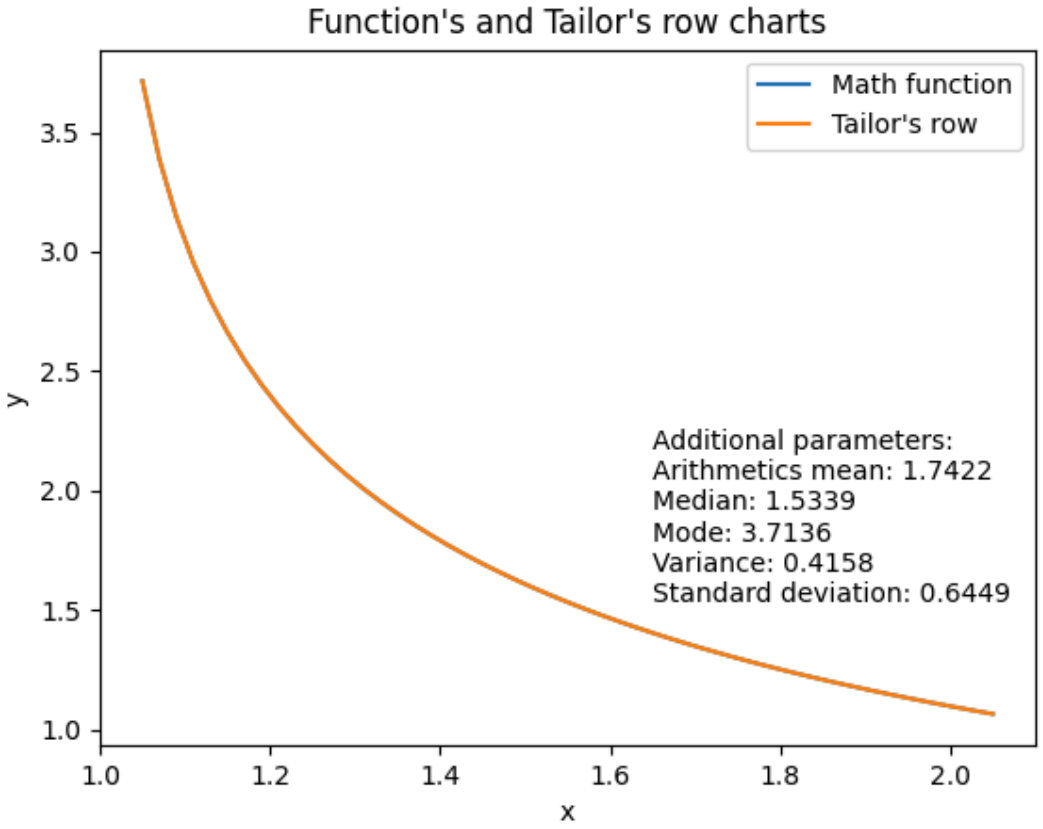
```
    def printBorders(self, type: str, length: int) -> None:
        ''' Method prints result table borders. '''
        for i in range(5):
            if i == 0:
                print("┌──" + "─" * length,
                      end="──┬──") if type == 'top' else print("\n├──" + "─" * length, end="──┼──") \
                    if type == 'mid' else print("\n└──" + "─" * length, end="──┴──")
            elif i != 4:
                print("─" * (length + 3), end="──┬──") if type == 'top' else print("─" * (length + 3), end="──┼──") \
                    if type == 'mid' else print("─" * (length + 3), end="──┴──")
            else:
                print("─" * (length + 3), end="──┐\n") if type == 'top' else print("─" * (length + 3), end="──┤\n") \
                    if type == 'mid' else print("─" * (length + 3), end="──┘\n")

    def displayConsole(self) -> None:
        ''' Method displays the result in console. '''
        for x in self.x:
            self.mathResults.append(self.tailor.countMath(x))
            tailorSum, iterations = self.tailor.countTailor(
                x, self.eps, self.mathResults[-1])
            self.tailorResults.append(tailorSum)
            self.iterations.append(iterations)
        length = max(len(str(max(self.tailorResults))), len(str(max(self.iterations))),
                     len((str(self.eps))), len(str(max(self.x))), 9)
        data = [["x", "n", "F(x)", "Math F(x)", "eps"]]
        for i in range(len(self.x)):
            data.append([self.x[i], self.iterations[i], round(self.tailorResults[i], len(str(self.eps))),
                         round(self.mathResults[i], len(str(self.eps))), self.eps])
        self.printBorders('top', length)
        for i in range(len(self.x) + 1):
            for j in range(5):
                if j == 0:
                    print("│  " + str(data[i][j]) + " " *
                          (length - len(str(data[i][j]))), end="  │  ",)
                else:
                    print(str(data[i][j]).format() + " " * (length + 3 -
                                                            len(str(data[i][j]))), end="  │  ")
            self.printBorders(
                'mid', length) if i != len(self.x) else self.printBorders('bot', length)
        print()
```

```
    def displayCharts(self):
        ''' Method creates function and row charts. '''
        plt.figure()
        plt.plot(self.x, self.mathResults, label='Math function')
        plt.plot(self.x, self.tailorResults, label='Tailor\'s row')
        plt.annotate(self.tailor.getAdditionalParams(),
                     (median(self.x) + 0.1, median(self.tailorResults) + 0.1))
        plt.legend()
        plt.title('Function\'s and Tailor\'s row charts')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.savefig('./task3/charts.png')

```

Пример использования:

```
PS D:\IGI\253502_KRASYOV_11\IGI\LR4> python main.py
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 3
 Task3
With a given accuracy calculate the value of a function using Tailor's row. Determine additional parameters: arithmetic mean, median, mode, dispersion, st
andard deviation of the sequence. Using the `matplotlib` library, draw charts of different colors on the same coordinate axis
Enter accuracy (0 < acc < 1) (0 -> Exit function): 1e-6
Enter start x (|x| > 1): 1.05
Enter end x (|x| > 1 & x > start): 2.05
Enter number of points (N > 0): 50
```

x	n	F(x)	Math F(x)	eps
1.05	138	3.71357	3.71357	1e-06
1.07	100	3.38681	3.38681	1e-06
1.09	79	3.14511	3.14511	1e-06
1.11	65	2.95396	2.95396	1e-06
1.13	56	2.79634	2.79634	1e-06



Задание 4. В соответствии с заданием своего варианта разработать базовые классы и классы наследники. Построить квадрат, на одной стороне которого, как на основании, построен равносторонний треугольник со стороной a.

Функция, запускающая задание

```python
@RunTask(Task4)
def runTask4(self, task: Task4) -> str | None:
    ''' Task4\nConstruct a square, on one side of which, as a base, an equilateral triangle with side a is built.
    '''
    return task()
```

Абстрактный класс `Геометрическая фигура`:

```python
from abc import ABC


class GeometricFigure(ABC):
    def area(self):
        ''' Method returns area of the figure. '''
        pass

```

Класс `Цвет фигуры`:

```python
class FigureColor():
    def __init__(self, color: str) -> None:
        self._color = color

    @property
    def color(self) -> str:
        ''' Square name getter. '''
        return self._color

```

Класс-примесь для классов квадрата и треугольника:

```python
class FigureMixin(object):
    def __str__(self) -> str:
        ''' Method returns figure info. '''
        return 'Name: {}\nColor: {}\nSide: {}\nArea: {}'.format(self.name, self.color.color, self.side, self.area())

```

Класс квадрата:

```python
from task4.geometricFigure import GeometricFigure
from task4.figureColor import FigureColor
from task4.figureMixin import FigureMixin


class Square(GeometricFigure, FigureMixin):
    def __init__(self, side: float, color: str) -> None:
        ''' Inits class object. '''
        self.side = side
        self.color = FigureColor(color)

    def area(self) -> float:
        ''' Method returns square area. '''
        return self.side**2

    @property
    def name(self) -> str:
        ''' Square name getter. '''
        return self._name

    @name.setter
    def name(self, value: str) -> None:
        ''' Square name setter. '''
        self._name = value.capitalize()
```

Класс треугольника:

```python
from task4.geometricFigure import GeometricFigure
from task4.figureColor import FigureColor
from task4.figureMixin import FigureMixin
from math import sqrt


class Triangle(GeometricFigure, FigureMixin):
    def __init__(self, side: float, color: str) -> None:
        ''' Inits class object. '''
        self.side = side
        self.color = FigureColor(color)

    def area(self) -> float:
        ''' Method returns triangle area. '''
        return self.side**2 * sqrt(3)/4

    @property
    def name(self) -> str:
        ''' Triangle name getter. '''
        return self._name

    @name.setter
    def name(self, value: str) -> None:
        ''' Triangle name setter. '''
        self._name = value.capitalize()

```

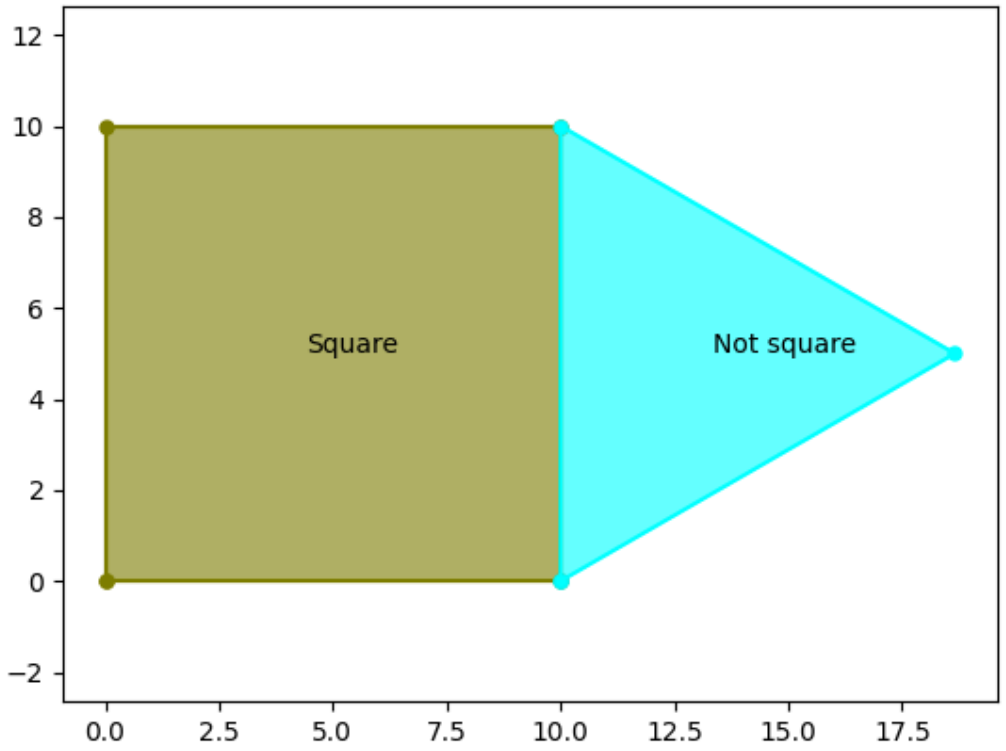
Общий класс для задания 4:

```python
from utils import validateNumberInput, validateColorInput
from task4.square import Square
from task4.triangle import Triangle, sqrt
from matplotlib import pyplot as plt


class Task4():
    def __init__(self) -> None:
        ''' Inits class object. '''
        self.square = None
        self.triangle = None

    def __call__(self) -> None | str:
        ''' Runs class methods. '''
        self.validateInput()
        if self.square is None:
            return 'stop'
        self.displayInfo()
        self.displayCharts()

    def validateInput(self) -> None:
        ''' Function for validation input of square side. '''
        side = validateNumberInput('length of square side (0 -> Exit function)', 'length must be bigger 0', float,
                                   condition=lambda x: x < 0)
        if side == 0:
            self.square = None
            return
        self.square = Square(side, validateColorInput('square'))
        self.triangle = Triangle(side, validateColorInput('triangle'))
        self.square.name = input('Enter square name: ')
        self.triangle.name = input('Enter triangle name: ')

    def displayInfo(self) -> None:
        ''' Method displays figures info in console. '''
        print('\nSquare info:')
        print(self.square())
        print('\nTriangle info:')
        print(self.triangle())
```

```python
    def displayCharts(self) -> None:
        ''' Method displays figures charts. '''
        side = self.square.side
        square_x = [0, side, side, 0, 0]
        square_y = [0, 0, side, side, 0]
        triangle_x = [side, side, (sqrt(3)*0.5 + 1)*side, side]
        triangle_y = [0, side, 0.5*side, 0]
        plt.figure()
        plt.xlim(0, (sqrt(3)*0.5 + 1)*side + 1)
        plt.ylim(0, side + 1)
        plt.axis('equal')
        plt.plot(square_x, square_y, color=self.square.color.color, marker='o',
                 markersize=5, linestyle='-')
        plt.plot(triangle_x, triangle_y, color=self.triangle.color.color, marker='o',
                 markersize=5, linestyle='-', label=self.triangle.name)
        plt.annotate(self.square.name, (side/2 -
                     len(self.square.name)*0.1, side/2))
        plt.annotate(self.triangle.name,
                     ((sqrt(3)*0.25 + 1)*side -
                      len(self.triangle.name)*0.1, side/2))
        plt.fill(square_x, square_y, color=self.square.color.color, alpha=0.6)
        plt.fill(triangle_x, triangle_y,
                 color=self.triangle.color.color, alpha=0.6)
        plt.autoscale()
        plt.savefig('./task4/charts.png')

```

Пример использования:

```
PS D:\IGI\253502_KRASYOV_11\IGI\LR4> python main.py
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 4
 Task4
Construct a square, on one side of which, as a base, an equilateral triangle with side a is built.
Enter length of square side (0 -> Exit function): 10
Input square color: olive
Input triangle color: cyan
Enter square name: Square
Enter triangle name: Not square

Square info:
Name: Square
Color: olive
Side: 10.0
Area: 100.0

Triangle info:
Name: Not square
Color: cyan
Side: 10.0
Area: 43.30127018922193
Do you want to finish task? (0 -> No | anything else -> Yes): yes
Task exit
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 0
Program finished!
PS D:\IGI\253502_KRASYOV_11\IGI\LR4>
```



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). 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).

Функция, запускающая задание:

```
42    @RunTask(Task5)
43    def runTask5(self, task: Task5) -> str | None:
44        ''' Task5\nGenerate an integer matrix A[n,m] using a random number generator.Find all elements that exceed the
          specified number B in absolute value. Count the number of such elements and write them to array C. Calculate
          the median value for this array C. '''
45        return task()
```

Класс, хранящий матрицу и анализирующий её:

```python
from generators import matrixGenerator
import numpy as np


class Matrix():
    def __call__(self, rows: int, columns: int) -> None:
        ''' Method generates matrix[rows, columns]. '''
        self.matrix = np.array(matrixGenerator(rows, columns))
        print('Generated matrix:')
        for row in self.matrix:
            print(row)
        print()

    def findElements(self, number: float):
        ''' Method returns numpy array with elements greater then `number`. '''
        return self.matrix[np.abs(self.matrix) > number]

    def findMedianNp(self, array):
        ''' Method counts array median. '''
        return np.median(array)

    def findMedian(self, array) -> float:
        ''' Method counts array median. '''
        array.sort()
        l = len(array)
        return array[l // 2] if l % 2 else (array[l // 2 - 1] + array[l // 2]) / 2

```

Общий класс для задания 5:

```python
from utils import validateNumberInput
from task5.matrix import Matrix


class Task5(Matrix):
    def __init__(self) -> None:
        ''' Inits class object. '''
        self.matrix = None

    def __call__(self) -> None | str:
        ''' Runs class methods. '''
        self.validateInput()
        if self.matrix is None:
            return 'stop'
        self.analiseMatrix()

    def validateInput(self) -> None:
        ''' Method validates number of matrix rows & columns input. '''
        rows = validateNumberInput('number of matrix rows (0 -> Exit function)',
                                   'number must be bigger 0', int, condition=lambda x: x < 0)
        if rows == 0:
            self.matrix = None
            return
        columns = validateNumberInput('number of matrix columns',
                                      'number must be bigger 0', int, condition=lambda x: x <= 0)
        super().__call__(rows, columns)

    def printArray(self, array, number: float):
        ''' Method returns numpy array of elements greater then `number`. '''
        print(f'Ammount of elements greater then |{number}|: {len(array)}')
        if len(array) > 0:
            print('Elements:')
            for el in array:
                print(el, end=" ")
        print()

    def analiseMatrix(self):
        ''' Method analises matrix. '''
        number = validateNumberInput('any number', '', float)
        C = self.findElements(number)
        self.printArray(C, number)
        if len(C) == 0:
            print('Array `C` is empty! Can\'t find median!')
            return
        print(f'NumPy median -> {self.findMedianNp(C)}')
        print(f'Programmed median -> {self.findMedian(C)}')

```

Пример использования:

```
PS D:\IGI\253502_KRASYOV_11\IGI\LR4> python main.py
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 5
 Task5
Generate an integer matrix A[n,m] using a random number generator.Find all elements that exceed the specified number B in absolute value. Count the number of such elements and write them to
array C. Calculate the median value for this array C.
Enter number of matrix rows (0 -> Exit function): 3
Enter number of matrix columns: 4
Generated matrix:
[  41   79 -145  138]
[ 57 127 125  37]
[ 44  19 -16 102]

Enter any number: 75
Ammount of elements greater then 75.0: 6
Elements:
79 -145 138 127 125 102
NumPy median -> 113.5
Programmed median -> 113.5
Do you want to finish task? (0 -> No | anything else -> Yes): нуы
Task exit
Enter number of task from 1 to 5, or 6 to perform data analysis, or 0 to exit: 0
Program finished!
PS D:\IGI\253502_KRASYOV_11\IGI\LR4>
```

Дополнительное задание. Используя любой из наборов данных (datasets) на ресурсе: https://www.kaggle.com/datasets исследовать основные возможности библиотеки Pandas.

Датасет - модели машин с основными характеристиками и ценой.

Класс для анализа данных:

```python
import pandas as pd
from sys import maxsize
from analysis.plotDrawer import PlotDrawer


class DataAnalisys(PlotDrawer):
    def __init__(self, path: str = './data.csv') -> None:
        ''' Inits class object. '''
        self.df = pd.read_csv(path)

    def __call__(self) -> None:
        ''' Method runs class methods. '''
        self.getInfo()
        print('Dataset analisys:\n')
        self.getYearDifference()
        self.getMileageDifference()
        self.getAvgMpg()
        self.drawYearPricePlot(self.df)
        self.drawMileagePricePlot(self.df)
        self.drawMpgPricePlot(self.df)

    def getInfo(self) -> None:
        ''' Method returns dataset info. '''
        print('Dataset:')
        print(self.df.info())
        print(self.df)

    def getAllModels(self, model: str, year: int = 1970, price: int = maxsize, fuelType: str = 'Petrol') -> None:
        ''' Method returns all cars satisfying release year (`year`), max price (`price`) & fuel type (`fuelType`). '''
        cars = self.df[(self.df['model'] == model) & (self.df['year'] >= year) &
                       (self.df['price'] <= price) & (self.df['fuelType'] == fuelType)]
        print(cars)

    def getAvgPrice(self, yearStart: int = 1970, yearEnd: int = 2024) -> float:
        ''' Method returns price mean of cars released between `yearStart` & `yearEnd`. '''
        cars = self.df[(self.df['year'] <= yearEnd) &
                       (self.df['year'] >= yearStart)]
        return cars['price'].mean()

    def getMedianTax(self) -> float:
        ''' Method returns median car tax. '''
        return self.df['tax'].median()
```

```python
    def getAvgMpg(self) -> None:
        ''' Method compares car\'s MPG depending on tax. '''
        print('MPG compare depending on tax:')
        tax = self.getMedianTax()
        print(f'Average tax: {tax}')
        print(
            f'Average MPG of cars with tax >= {tax}: {self.df[self.df["tax"] >= tax]["mpg"].mean():.2f}')
        print(
            f'Average MPG of cars with tax < {tax}: {self.df[self.df["tax"] < tax]["mpg"].mean():.2f}\n')

    def getYearDifference(self, year: int = 2019):
        ''' Method compares car\'s price depending on release year. '''
        print('Price compare depending on release year:')
        p1, p2 = self.getAvgPrice(
            yearEnd=year), self.getAvgPrice(yearStart=year)
        print(f'Average price before {year}:  ${p1:,.0f}')
        print(f'Average price after {year}:  ${p2:,.0f}')
        print(f'Ratio (after {year} / before {year}): {p2/p1:.2f}\n')

    def getMileageDifference(self):
        ''' Method compares car\'s price depending on mileage'''
        print('Price compare depending on car mileage:')
        maxMil = self.df.iloc[self.df['mileage'].idxmax()]
        print('Max mileage car:')
        self.printModelInfo(maxMil)
        minMil = self.df.iloc[self.df['mileage'].idxmin()]
        print('Min mileage car:')
        self.printModelInfo(minMil)
        print(
            f"Price difference: {minMil['price']} - {maxMil['price']} = {minMil['price'] - maxMil['price']}\n")

    def printModelInfo(self, car) -> None:
        ''' Method returns car info. '''
        print(f'Model: {car["model"]}')
        print(f'Year: {car["year"]}')
        print(f'Price: {car["price"]}')
        print(f'Transmission: {car["transmission"]}')
        print(f'Mileage: {car["mileage"]}')
        print(f'FuelType: {car["fuelType"]}')
        print(f'Tax: {car["tax"]}')
        print(f'MPG: {car["mpg"]}')
        print(f'EngineSize: {car["engineSize"]}')
        print(f'Manufacturer: {car["Manufacturer"]}\n')

```

Класс, выполняющий построение графиков по данным:

```python
from matplotlib import pyplot as plt


class PlotDrawer():
    def drawYearPricePlot(self, df):
        ''' Method draws average car price chart depending on year. '''
        years, prices = [], []
        for year in range(1995, 2025):
            cars = df[df['year'] == year]
            if len(cars) > 0:
                years.append(year)
                prices.append(cars['price'].mean())
        plt.figure()
        plt.title('Average car price 1995-2024')
        plt.plot(years, prices, marker='o', markersize=2, linestyle='-')
        plt.xlabel('Years')
        plt.ylabel('Average price')
        plt.savefig('./analysis/year-price.png')

    def drawMileagePricePlot(self, df):
        ''' Method draws average car price chart depending on mileage '''
        mileages, prices = [], []
        for m in range(1, 41):
            cars = df[(df['mileage'] < int(
                1e4*m)) & (df['mileage'] > int(1e4*(m - 1)))]
            if (len(cars) == 0):
                continue
            mileages.append(1e4*m)
            prices.append(cars['price'].mean())
        plt.figure()
        plt.title('Average car price depending on mileage')
        plt.plot(mileages, prices, marker='o', markersize=2, linestyle='-')
        plt.xlabel('Mileages')
        plt.ylabel('Average price')
        plt.savefig('./analysis/mileage-price.png')

    def drawMpgPricePlot(self, df):
        ''' Method draws average car price chart depending on mpg '''
        mpgs, prices = [], []
        for i in range(10, 480, 10):
            cars = df[(df['mpg'] < i) & (df['mpg'] > i - 10)]
            if len(cars) == 0:
                continue
            mpgs.append(i)
            prices.append(cars['price'].mean())
        plt.figure()
        plt.title('Average car price depending on mpg')
        plt.plot(mpgs, prices, marker='o', markersize=2, linestyle='-')
        plt.xlabel('MPG')
        plt.ylabel('Average price')
        plt.savefig('./analysis/mpg-price.png')

```

Пример использования:

```
from analysis import DataAnalisys
ds = DataAnalisys()
ds.df.head()
ds.df
```

[1]

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	Manufacturer
0	I10	2017	7495	Manual	11630	Petrol	145	60.1	1.0	hyundi
1	Polo	2017	10989	Manual	9200	Petrol	145	58.9	1.0	volkswagen
2	2 Series	2019	27990	Semi-Auto	1614	Diesel	145	49.6	2.0	BMW
3	Yeti Outdoor	2017	12495	Manual	30960	Diesel	150	62.8	2.0	skoda
4	Fiesta	2017	7999	Manual	19353	Petrol	125	54.3	1.2	ford
...	...	...	...	...	...	...	...	...	...	...
97707	Fiesta	2017	10447	Automatic	8337	Petrol	145	54.3	1.0	ford
97708	3 Series	2014	14995	Manual	25372	Diesel	30	61.4	2.0	BMW
97709	Fiesta	2017	8950	Manual	19910	Petrol	125	54.3	1.2	ford
97710	Astra	2017	10700	Automatic	24468	Petrol	125	50.4	1.4	vauxhall
97711	Grandland X	2019	15798	Manual	10586	Diesel	150	48.7	1.5	vauxhall

97712 rows × 10 columns

```
ds.df.info()
```

[2]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97712 entries, 0 to 97711
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   model         97712 non-null  object
 1   year          97712 non-null  int64
 2   price         97712 non-null  int64
 3   transmission  97712 non-null  object
 4   mileage       97712 non-null  int64
 5   fuelType      97712 non-null  object
 6   tax           97712 non-null  int64
 7   mpg           97712 non-null  float64
 8   engineSize    97712 non-null  float64
 9   Manufacturer  97712 non-null  object
dtypes: float64(2), int64(4), object(4)
memory usage: 7.5+ MB
```

```
cars = ds.df[(ds.df['model'] == ' A4') & (ds.df['year'] >= 2017) &
             (ds.df['price'] <= 20000)]
ds.df.head()
cars
```

[3]

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	Manufacturer
28	A4	2017	18182	Manual	18574	Petrol	145	51.4	1.4	Audi
569	A4	2017	15540	Automatic	35452	Diesel	145	70.6	2.0	Audi
611	A4	2017	18990	Semi-Auto	30417	Diesel	30	62.8	2.0	Audi
672	A4	2018	18995	Semi-Auto	18696	Petrol	145	50.4	1.4	Audi
1108	A4	2017	17190	Manual	35790	Diesel	30	67.3	2.0	Audi
...	...	...	...	...	...	...	...	...	...	...
96842	A4	2017	15441	Manual	49000	Diesel	30	64.2	2.0	Audi
97030	A4	2017	17445	Manual	36187	Diesel	20	70.6	2.0	Audi
97313	A4	2017	17490	Manual	35000	Petrol	125	49.6	2.0	Audi
97393	A4	2018	17495	Semi-Auto	9296	Petrol	150	53.3	1.4	Audi
97395	A4	2017	20000	Automatic	20852	Diesel	30	65.7	2.0	Audi

306 rows × 10 columns

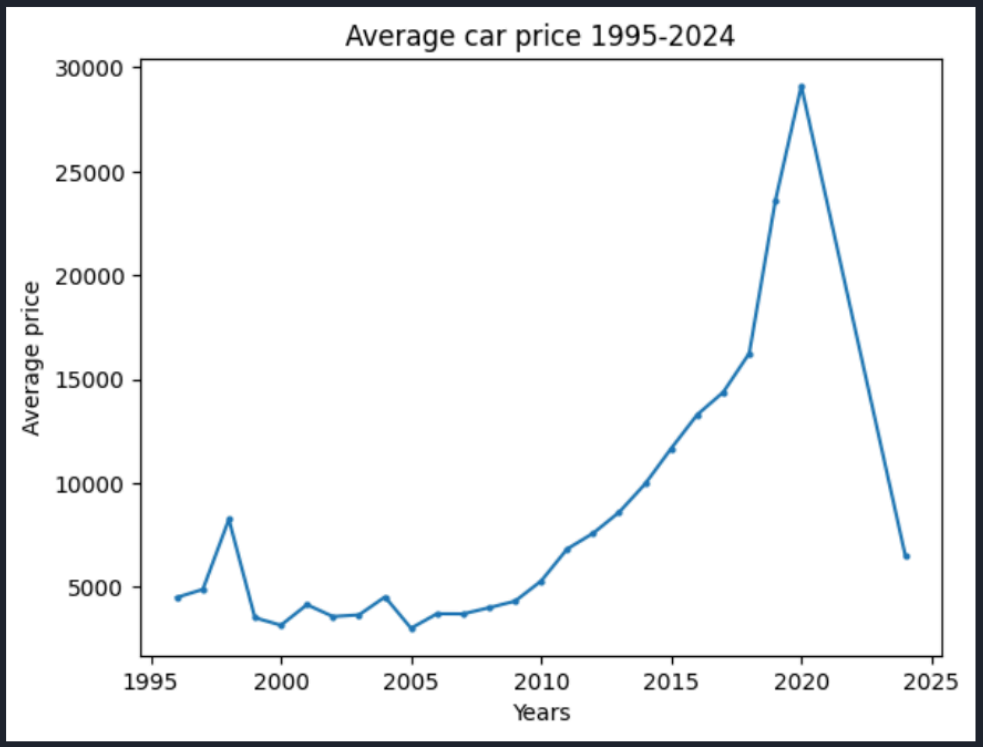
+ Code + Markdown

```
ds.getYearDifference()
```

[4]

```
Price compare depending on release year:
Average price before 2019:  $16,242
Average price after 2019:  $24,310
Ratio (after 2019 / before 2019): 1.50
```

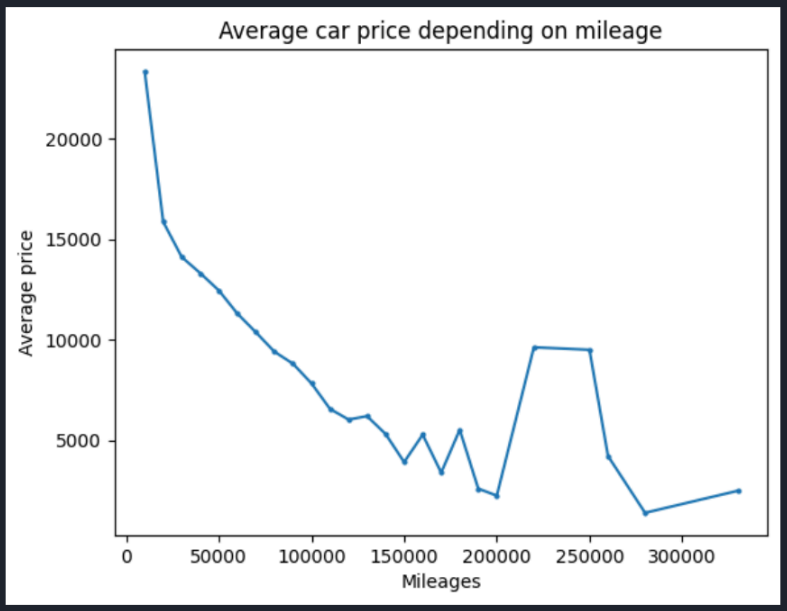
```
[5]  ds.drawYearPricePlot(ds.df)
```




```
[6]  ds.getMileageDifference()
```

```
Price compare depending on car mileage:
Max mileage car:
Model:  A6
Year: 2008
Price: 2490
Transmission: Manual
Mileage: 323000
FuelType: Diesel
Tax: 200
MPG: 44.1
EngineSize: 2.0
Manufacturer: Audi

Min mileage car:
Model:  C Class
Year: 2019
Price: 26995
Transmission: Semi-Auto
Mileage: 1
FuelType: Diesel
Tax: 145
MPG: 54.3
EngineSize: 1.6
Manufacturer: merc

Price difference: 26995 - 2490 = 24505
```

```
[7]
ds.drawMileagePricePlot(ds.df)
```




```
[10]
ds.getAvgMpg()
```

```
MPG compare depending on tax:
Average tax: 145.0
Average MPG of cars with tax >= 145.0: 50.49
Average MPG of cars with tax < 145.0: 65.29
```

```
[11]
ds.drawMpgPricePlot(ds.df)
```