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: Трошко Александр Олегович

Группа: 253503

errors.py

```python
2 usages
def is_command(value: str) -> int:
    """Checks the value between 0 and 6. """
    while True:
        try:
            value = int(value)
            if 0 < value < 6:
                return value
            value = input("Value should be between 0 and 6, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid input: ")

```

menu.py

```python
2 usages
def menu():
    """Returns menu. """
    print("\n1: Start Task1")
    print("2: Start Task2")
    print("3: Start Task3")
    print("4: Start Task4")
    print("5: Start Task5")
    print("6: Exit")

```

requirements.txt

```
matplotlib==3.8.4
pandas==2.2.2
numpy==1.26.4
prettytable==3.10.0
```

main.py

```python
from errors import is_command
from menu import menu
import Task1
import Task2
import Task3
import Task4
import Task5


1 usage
def program():
    while True:
        menu()
        command = is_command(input("\nEnter value: "))
        if command == 1:
            Task1.main()
        if command == 2:
            Task2.main()
        if command == 3:
            Task3.main()
        if command == 4:
            Task4.main()
        if command == 5:
            Task5.main()
        if command == 6:
            break


if __name__ == "__main__":
    program()

```

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

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

Task1.py

```python
import csv
import pickle


1 usage
class Tree:
    def __init__(self, tree_type: str, total: int, healthy: int) -> None:
        if healthy > total:
            raise ValueError("There cannot be more healthy trees than the total number")
        self._tree_type = tree_type
        self._total = total
        self._healthy = healthy

    def __str__(self) -> str:
        return f"Type: {self.tree_type}, Total: {self.total}, Healthy: {self.healthy}"

    5 usages (4 dynamic)
    @property
    def tree_type(self):
        return self._tree_type

    6 usages (3 dynamic)
    @property
    def total(self):
        return self._total

    4 usages (2 dynamic)
    @property
    def healthy(self):
        return self._healthy

    3 usages (2 dynamic)
    @property
    def diseased(self):
        return self._total - self._healthy

    @property
    def diseased_percent(self):
        return (self.diseased - self.total) * 100 if self.total else 0
```

Task1.py

```
35
        2 usages (2 dynamic)
36      @healthy.setter
37      def healthy(self, value: int):
38          if value > self._healthy:
39              raise ValueError("There cannot be more healthy trees than the total number")
40          self._healthy = value
41
42
    5 usages
43  class Forest:
44      def __init__(self, trees: dict):
45          self._trees = list()
46          self.total_count = 0
47          self.total_healthy_count = 0
48          for tree_type, info in trees.items():
49              self.add_tree(tree_type, info["total"], info["healthy"])
50
51      @property
52      def trees(self):
53          return self._trees
54
        1 usage
55      def add_tree(self, tree_type: str, total: int, healthy: int) -> None:
56          """Add a tree to the trees list. """
57          if healthy > total:
58              raise ValueError("There cannot be more healthy trees than the total number")
59          self._trees.append(Tree(tree_type, total, healthy))
60          self.total_healthy_count += healthy
61          self.total_count += total
62
        1 usage
63      def total_trees_count(self) -> str:
64          """Return the total number of trees in the trees list. """
65          return f"\nTotal trees in forest: {self.total_count}"
66
        1 usage
67      def total_healthy_trees_count(self) -> str:
68          """Return the healthy number of trees in the trees list. """
69          return f"\nTotal healthy trees in forest: {self.total_healthy_count}"
```

Task1.py

```
    def get_tree(self, tree_type: str) -> str:
        """Return the info of the tree in the trees list. """
        tree_info = list()
        for tree in self._trees:
            if tree.tree_type == tree_type:
                tree_info.append(f"\nType: {tree.tree_type}, Total: {tree.total}, Healthy: {tree.healthy}")
                return "\n".join(tree_info)
        return f"\nNo tree with type '{tree_type}'"

    def get_all_trees(self) -> str:
        """Return the all trees in the trees list. """
        return "\n".join(
            [f"Type: {tree.tree_type}, Total: {tree.total}, Healthy: {tree.healthy}" for tree in self._trees])

    1 usage
    def relative_diseased_percent(self) -> str:
        """Return the relative diseased percentage of the trees in the trees list. """
        return (f"\nRelative abundance (%) of diseased trees: "
                f"{(sum(tree.diseased for tree in self._trees) / self.total_count) * 100.0}")\
            if self.total_count > 0 else 0

    1 usage
    def relative_diseased_percent_by_type(self) -> str:
        """Return the relative diseased percentage by type of the trees in the trees list. """
        return "\n".join(
            [f"Relative abundance (%) of diseased '{tree.tree_type}': "
             f"{round(tree.diseased / tree.total * 100.0, 2)}" for tree in self._trees])


2 usages
class FileMixin:
    def write_to_file(self, data: dict, file_name) -> None:
        """Write the data to the specified file. """
        raise NotImplementedError

    def read_from_file(self, file_name) -> None:
        """Read the data from the specified file. """
        raise NotImplementedError
```

Task1.py

```python

1 usage
class CSVSerializer(FileMixin):
    1 usage
    def write_to_file(self, data: dict, file_name) -> None:
        """Write data to file. """
        with open(file_name, "w", newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['type', 'total', 'healthy'])
            for tree_type, info in data.items():
                writer.writerow([tree_type, info['total'], info['healthy']])

    1 usage
    def read_from_file(self, file_name) -> 'Forest':
        """Read data from file. """
        forests = dict()
        with open(file_name, "r", newline='') as file:
            reader = csv.DictReader(file)
            for row in reader:
                forests[row['type']] = {"total": int(row['total']), "healthy": int(row['healthy'])}
        return Forest(forests)


1 usage
class PickleSerializer(FileMixin):
    1 usage
    def write_to_file(self, data: dict, file_name) -> None:
        """Write data to file. """
        with open(file_name, "wb") as file:
            pickle.dump(data, file)

    1 usage
    def read_from_file(self, file_name) -> 'Forest':
        """Read data from file. """
        with open(file_name, "rb") as file:
            return Forest(pickle.load(file))


```

Task1.py

```python
def is_command(value: str) -> int:
    """Checks the value to be between 0 and 11. """
    while True:
        try:
            value = int(value)
            if 0 < value < 11:
                return value
            value = input("Value should be between 0 and 11, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")


1 usage
def menu() -> None:
    """Menu for user input. """
    print("\n1: Writing to a file using CSV")
    print("2: Reading from a file using CSV")
    print("3: Writing to a file using pickle")
    print("4: Reading from a file using pickle")
    print("5: Display the total number of trees in the control area")
    print("6: Display the total number of healthy trees")
    print("7: Display the relative abundance (%) of diseased trees")
    print("8: Display the relative abundance (%) different species, including patients (%) for each species")
    print("9: Display the information about the type of tree entered from the keyboard")
    print("10: Exit\n")
```

Task1.py

```python

1 usage
def main() -> None:
    data = {
        "Pine": {"total": 500, "healthy": 100},
        "Ash": {"total": 110, "healthy": 100},
        "Oak": {"total": 50, "healthy": 10}
    }
    forest = Forest(data)
    csv_serializer = CSVSerializer()
    pickle_serializer = PickleSerializer()
    while True:
        menu()
        command = is_command(input("Enter value: "))
        if command == 1:
            csv_serializer.write_to_file(data, file_name: "forest.csv")
        if command == 2:
            forest = csv_serializer.read_from_file("forest.csv")
        if command == 3:
            pickle_serializer.write_to_file(data, file_name: "forest.txt")
        if command == 4:
            forest = pickle_serializer.read_from_file("forest.txt")
        if command == 5:
            print(forest.total_trees_count())
        if command == 6:
            print(forest.total_healthy_trees_count())
        if command == 7:
            print(forest.relative_diseased_percent())
        if command == 8:
            print(forest.relative_diseased_percent_by_type())
        if command == 9:
            print(forest.get_tree(input("\nEnter the tree: ")))
        if command == 10:
            break

```

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

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

- количество предложений в тексте;
- количество предложений в тексте каждого вида отдельно (повествовательные, вопросительные и побудительные);
- среднюю длину предложения в символах (считаются только слова);

– среднюю длину слова в тексте в символах;
– количество смайликов в заданном тексте. Смайликом будем считать последовательность символов, удовлетворяющую условиям:

первым символом является либо «;» (точка с запятой) либо «:» (двоеточие) ровно один раз;

далее может идти символ «-» (минус) сколько угодно раз (в том числе символ минус может идти ноль раз);

в конце обязательно идет некоторое количество (не меньше одной) одинаковых скобок из следующего набора: «(», «)», «[», «]»;

внутри смайлика не может встречаться никаких других символов. Например, эта последовательность является смайликом: «;---------[[[[[[[[». Эти последовательности смайликами не являются: «]», «;--»,«:»,«()».

Вар-т	Условие
1.	Вывести все слова, начинающиеся со строчной согласной буквы. Определить, является ли последовательность букв корректным автомобильным номером определить, сколько слов имеют минимальную длину; вывести все слова, за которыми следует запятая; найти самое длинное слово, которое оканчивается на 'y'

Task2.py

```python
import re
from zipfile import ZipFile


1 usage
class TextAnalyzer:
    def __init__(self, filename: str) -> None:
        self.__text = None
        self.__filename = filename

    @property
    def text(self) -> str:
        return self.__text

    @property
    def filename(self) -> str:
        return self.__filename

    2 usages
    def read_file(self, filename: str) -> str:
        """Read text from a file. """
        with open(filename, 'r') as file:
            self.__text = file.read()
        return self.__text
```

Task2.py

```python
    def write_result_file(self, filename: str) -> None:
        """Write text to a file. """
        t = self.read_file("text.txt")
        with open(filename, 'w') as file:
            file.write(f"Count sentences in text: {self.get_count_sentences(t)}")
            file.write(f"\nCount sentences in text of each type: ")
            for keys, values in self.get_type_sentences(t).items():
                file.write(f"\n{keys}: {values}")
            file.write(f"\nAverage sentence length in characters: ")
            for keys, values in self.avg_sentence_length(t).items():
                file.write(f"\n{keys}: {values}")
            file.write(f"\nAverage word length in text in characters: {self.avg_word_length(t)}")
            file.write(f"\nCount of emoticons in a text: {self.search_smile(t)}")
            file.write(f"\nAll words starting with a lowercase consonant: {self.search_word(t)}")
            file.write(f"\nCar numbers: {self.search_car_number(t)}")
            file.write(f"\nWords that have a minimum length: {self.search_min_word(t)}")
            file.write(f"\nAll words followed by a comma: {self.search_comma(t)}")
            file.write(f"\nLongest word that ends with 'y': {self.search_y(t)}")

    1 usage
    def output(self) -> None:
        """Print text to console. """
        t = self.read_file("text.txt")
        print(f"\nCount sentences in text: {self.get_count_sentences(t)}")
        print(f"\nCount sentences in text of each type:")
        for keys, values in self.get_type_sentences(t).items():
            print(f"\t{keys}: {values}")
        print(f"\nAverage sentence length in characters: ")
        for keys, values in self.avg_sentence_length(t).items():
            print(f"\t{keys}: {values}")
        print(f"\nAverage word length in text in characters: {self.avg_word_length(t)}")
        print(f"\nCount of emoticons in a text: {self.search_smile(t)}")
        print(f"\nAll words starting with a lowercase consonant: {self.search_word(t)}")
        print(f"\nCar numbers: {self.search_car_number(t)}")
        print(f"\nWords that have a minimum length: {self.search_min_word(t)}")
        print(f"\nAll words followed by a comma: {self.search_comma(t)}")
        print(f"\nLongest word that ends with 'y': {self.search_y(t)}")
```

Task2.py

```python
    @staticmethod
    def write_zip(filename: str) -> None:
        """Write zip file. """
        with ZipFile( file: "results.zip",  mode: 'w') as myzip:
            myzip.write(filename)

    1 usage
    @staticmethod
    def info_zip(filename: str) -> None:
        """Print info about zip file. """
        with ZipFile(filename,  mode: 'r') as myzip:
            for item in myzip.infolist():
                print(f"\nFile Name: {item.filename} Date: {item.date_time} Size: {item.file_size}")

    @staticmethod
    def read_zip() -> None:
        """Read zip file. """
        with ZipFile( file: "results.zip",  mode: 'r') as myzip:
            print(myzip.read("results.txt"))

    2 usages
    @staticmethod
    def get_count_sentences(text: str) -> int:
        """Count the sentences number of text. """
        return len(re.findall( pattern: r'\w+[.!?]', text))

    2 usages
    @staticmethod
    def get_type_sentences(text: str) -> dict:
        """Count the type of sentences number of text. """
        patterns = dict()
        narrative_pattern = len(re.findall( pattern: r'\w+\.', text))
        question_pattern = len(re.findall( pattern: r'\w+\?', text))
        exclamation_pattern = len(re.findall( pattern: r'\w+!', text))
        patterns["narrative"] = narrative_pattern
        patterns["question"] = question_pattern
        patterns["exclamation"] = exclamation_pattern
        return patterns
```

Task2.py

```python
    @staticmethod
    def avg_sentence_length(text: str) -> dict:
        """Average sentence length of text. """
        text_pattern = [sentence.strip() for sentence in re.split( pattern: r'[.!?]', text) if sentence.strip()]
        avg_sentence_length = dict()
        for sentence in text_pattern:
            words_pattern = re.findall( pattern: r'\w+', sentence)
            total_characters = sum(len(word) for word in words_pattern if re.findall( pattern: r'[A-Za-z]+', word))
            total_words = sum(1 for word in words_pattern if re.findall( pattern: r'[A-Za-z]+', word))
            if total_words > 0:
                avg = total_characters / total_words
                avg_sentence_length[sentence] = avg
        return avg_sentence_length

    2 usages
    @staticmethod
    def avg_word_length(text: str) -> float:
        """Average word length of text. """
        word_pattern = [word for word in re.findall( pattern: r'\w+', text)]
        total_characters = sum(len(word) for word in word_pattern)
        if len(word_pattern) > 0:
            avg = total_characters / len(word_pattern)
            return avg
        return 0

    2 usages
    @staticmethod
    def search_smile(text: str) -> int:
        """Search for SMILES in text. """
        return len([word for word in re.findall( pattern: r'[:;]-*\(+|[:;]-*\)+|[:;]-*\[+|[:;]-*]+', text)])

    2 usages
    @staticmethod
    def search_word(text: str) -> list:
        """Search for words in text. """
        return [word for word in re.findall( pattern: r'[BCDFGHJKLMNPQRSTVWXYZ]\w*', text)]
```

Task2.py

```python
    @staticmethod
    def search_car_number(text: str):
        """Search for car numbers in text. """
        number_pattern = re.findall( pattern: r'[0-9]{4}\s[ABEIKMHOPCTX]{2}-[1-7]', text)
        if number_pattern:
            return number_pattern[:]
        return None

    2 usages
    @staticmethod
    def search_min_word(text: str) -> int:
        """Search for min word in text. """
        word_pattern = [word for word in re.findall( pattern: r'\w+', text)]
        return sum(1 for word in word_pattern if len(word) == min(len(word) for word in word_pattern))

    2 usages
    @staticmethod
    def search_comma(text: str) -> list:
        """Search for comma in text. """
        return [word.replace(',', '').strip() for word in re.findall( pattern: r'\w+,\s*', text)]

    2 usages
    @staticmethod
    def search_y(text: str) -> str:
        """Search for y value in text. """
        return ''.join(max([word for word in re.findall( pattern: r'\w+y\s', text)], key=lambda x: len(x)))


1 usage
def is_command(value: str) -> int:
    """Checks the value to be between 0 and 5. """
    while True:
        try:
            value = int(value)
            if 0 < value < 5:
                return value
            value = input("Value should be between 0 and 5, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")

```

```python
def menu() -> None:
    """Menu for user input. """
    print("\n1: Read text from file")
    print("2: Start the task")
    print("3: Get the info about zip")
    print("4: Exit\n")


1 usage
def main() -> None:
    txt = None
    while True:
        menu()
        command = is_command(input("Enter value: "))
        if command == 1:
            txt = TextAnalyzer("text.txt")
        if command == 2:
            if txt is None:
                print("\nText not found, please try again")
                continue
            txt.output()
            txt.write_result_file("results.txt")
            txt.write_zip("results.txt")
        if command == 3:
            if txt is None:
                print("\nText not found, please try again")
                continue
            txt.info_zip("results.zip")
        if command == 4:
            break

```

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.

x	n	$F(x)$	$Math\ F(x)$	eps

Здесь x – значение аргумента, F(x) – значение функции, n – количество просуммированных членов ряда, Math F(x) – значение функции, вычисленное с помощью модуля math.

в) сохранить графики в файл

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

Task3.py

```python
import math
import matplotlib.pyplot as plt
import statistics
from prettytable import PrettyTable


1 usage
class Statistics:
    def __init__(self) -> None:
        self._values = []
        self._series_results = []
        self._math_results = []

    10 usages
    @property
    def values(self) -> list:
        return self._values

    2 usages
    @property
    def series_results(self) -> list:
        return self._series_results

    2 usages
    @property
    def math_results(self) -> list:
        return self._math_results

    1 usage
    def add_data(self, value: float, series_result: float, math_result: float) -> None:
        """Adds data to the lists"""
        self.values.append(value)
        self.series_results.append(series_result)
        self.math_results.append(math_result)

    1 usage
    def mean(self) -> float:
        """Calculate the mean of the values."""
        return statistics.mean(self.values)
```

Task3.py

```python
    def median(self) -> float:
        """Calculate the median of the values. """
        return statistics.median(self.values)

    1 usage
    def mode(self) -> float:
        """Calculate the mode of the values. """
        return statistics.mode(self.values)

    1 usage
    def variance(self) -> float:
        """Calculate the variance of the values. """
        if len(self.values) < 2:
            return float('nan')
        return statistics.variance(self.values)

    1 usage
    def std(self) -> float:
        """Calculate the standard deviation of the values. """
        if len(self.values) < 2:
            return float('nan')
        return statistics.stdev(self.values)
```

Task3.py

```
    def plot_results(self) -> None:
        """Plot the results of the calculations. """
        plt.figure(figsize=(10, 5))
        plt.plot( *args: self.values, self.series_results, 'bo-', label='Taylor Series Approximation')
        plt.plot( *args: self.values, self.math_results, 'r', label='Math Module Computation')
        plt.xlabel('x')
        plt.ylabel('F(x)')
        plt.title('Function Approximations')
        plt.axhline( y: 0, color='black', linewidth=0.5)
        plt.axvline( x: 0, color='black', linewidth=0.5)
        plt.legend()
        plt.grid(True)
        plt.annotate( text: f'Mean: {round(self.mean())}', xy=(0.05, 0.95), xycoords='axes fraction', fontsize=12,
                     horizontalalignment='left', verticalalignment='top')
        plt.annotate( text: f'Median: {round(self.median())}', xy=(0.05, 0.90), xycoords='axes fraction', fontsize=12,
                     horizontalalignment='left', verticalalignment='top')
        plt.annotate( text: f'Mode: {round(self.mode())}', xy=(0.05, 0.85), xycoords='axes fraction', fontsize=12,
                     horizontalalignment='left', verticalalignment='top')
        variance_value = self.variance()
        if not math.isnan(variance_value):
            plt.annotate( text: f'Variance: {round(variance_value)}', xy=(0.05, 0.80), xycoords='axes fraction', fontsize=12,
                         horizontalalignment='left', verticalalignment='top')
        else:
            plt.annotate( text: 'Variance: Not computable', xy=(0.05, 0.80), xycoords='axes fraction', fontsize=12,
                         horizontalalignment='left', verticalalignment='top')
        std_value = self.std()
        if not math.isnan(std_value):
            plt.annotate( text: f'Std: {round(std_value)}', xy=(0.05, 0.75), xycoords='axes fraction', fontsize=12,
                         horizontalalignment='left', verticalalignment='top')
        else:
            plt.annotate( text: 'Std: Not computable', xy=(0.05, 0.75), xycoords='axes fraction', fontsize=12,
                         horizontalalignment='left', verticalalignment='top')
        plt.savefig('function_approximations.png')
        plt.show()


table = PrettyTable()
stat = Statistics()
```

Task3.py

```python
def is_size(value) -> int:
    """ Checks the value to make sure it is greater than 0. """
    while True:
        try:
            value = int(value)
            if value > 0:
                return value
            value = input("Value should be greater than 0, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")


2 usages
def is_eps(value) -> float:
    """ Checks the value to be between 0 and 1. """
    while True:
        try:
            value = float(value)
            if 0 < value < 1:
                return value
            value = input("Value should be between 0 and 1, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")


2 usages
def is_value(value) -> float:
    """ Checks the value to make sure it is greater than 1. """
    while True:
        try:
            value = float(value)
            if value > 1:
                return value
            value = input("Value should be greater than 1, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")
```

Task3.py

```python
def is_command(value) -> int:
    """ Checks the value to be between 0 and 4. """
    while True:
        try:
            value = int(value)
            if 0 < value < 4:
                return value
            value = input("Value should be between 0 and 4, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")


1 usage
def get_size_tuple() -> int:
    """ Returns a number that will be the size of the list. """
    return is_size(input("Enter size of list: "))


1 usage
def generator(size: int) -> tuple:
    """ Returns a sequence of numbers. """
    for _ in range(size):
        yield is_value(input("Enter values: "))


1 usage
def get_list(size: int) -> tuple:
    """ Returns eps and generator. """
    return is_eps(input("Enter eps: ")), tuple(generator(size))


1 usage
def get_values() -> tuple:
    """ Returns eps and value. """
    return is_eps(input("Enter eps: ")), is_value(input("Enter value: "))
```

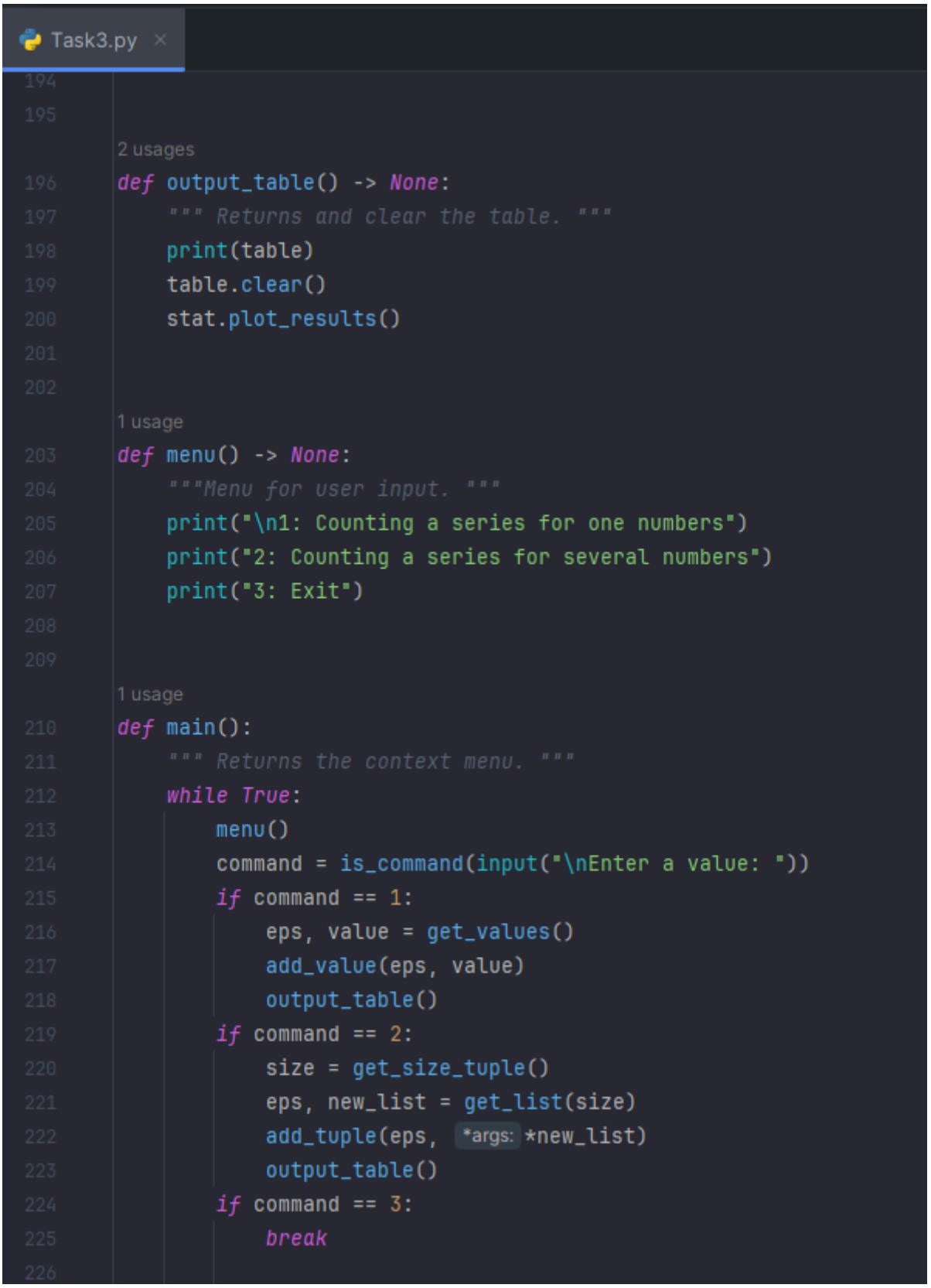
Task3.py

```python
def get_taylor_series_math(value: int) -> float:
    """ Returns the value of a function using a module math. """
    return math.log((value + 1) / (value - 1))


1 usage
def get_taylor_series(eps: float, value: int) -> tuple:
    """ Returns the value of a function using eps. """
    s = n = 0
    a = value
    while abs(a) > eps and n < 501:
        s += a
        a = 1 / ((2 * n + 1) * value ** (2 * n + 1))
        n += 1
    s -= value
    return n, 2 * s


2 usages
def add_value(eps: float, value) -> None:
    """ Adds values to the table. """
    n, s = get_taylor_series(eps, value)
    smath = get_taylor_series_math(value)
    stat.add_data(value, s, smath)
    table.field_names = ["x", "n", "F(x)", "Math F(x)", "eps"]
    table.add_row([value, n, s, smath, eps])


1 usage
def add_tuple(eps: float, *args) -> None:
    """ Unpacks the tuple and calls the method 'add_value'. """
    for value in args:
        add_value(eps, value)
```



Задание 4. В соответствии с заданием своего варианта разработать базовые классы и классы наследники.

Требования по использованию классов:

Абстрактный класс «Геометрическая фигура» содержит абстрактный метод для вычисления площади фигуры (https://docs.python.org/3/library/abc.html)

Класс «Цвет фигуры» содержит свойство для описания цвета геометрической фигуры (https://docs.python.org/3/library/functions.html#property)

Класс «Прямоугольник» (Круг, Ромб, Квадрат, Треугольник и т.д.) наследуется от класса «Геометрическая фигура». Класс должен содержать конструктор по параметрам «ширина», «высота» (для другого типа фигуры соответствующие параметры, например, для круга задаем «радиус») и «цвет». В конструкторе создается объект класса «Цвет фигуры» для хранения цвета. Класс должен переопределять метод, вычисляющий площадь фигуры https://docs.python.org/3/library/math.html .

Для класса «Прямоугольник»(тип фигуры в инд. задании)

определить метод, который возвращает в виде строки основные параметры фигуры, ее цвет и площадь. Использовать метод format (https://pyformat.info/)

название фигуры должно задаваться в виде поля данных класса и возвращаться методом класса.

В корневом каталоге проекта создайте файл main.py для тестирования классов. Используйте конструкцию, описанную в https://docs.python.org/3/library/__main__.html

Пример объекта: Прямоугольник синего цвета шириной 5 и высотой 8.

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.

Вар	Условие
1.	Построить ромб по стороне a и острому углу R (в градусах).

Task4.py

```python
from abc import ABC, abstractmethod
import math


1 usage
class GeometricFigure(ABC):
    @abstractmethod
    def calculate_area(self):
        pass


1 usage
class Color:
    def __init__(self, color):
        self._color = color

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


1 usage
class Rhombus(GeometricFigure):
    figure = "Ромб"

    def __init__(self, side, angle, color):
        self.color = Color(color)
        self._side = side
        self._angle = angle

    2 usages
    def calculate_area(self):
        """Calculate the area of rhombus. """
        return self._side ** 2 * math.sin(math.radians(self._angle))
```

Task4.py

```
32      def get_info(self):
33          """Print info about the figure. """
34          return "{type} {color} цвета со стороной {side} и углом {angle} градусов".format(
35              type=self.figure,
36              color=self.color.color,
37              side=self._side,
38              angle=self._angle
39          )
40
41
        1 usage
42  def is_command(value: str) -> int:
43      """Checks the value to be between 0 and 4. """
44      while True:
45          try:
46              value = int(value)
47              if 0 < value < 4:
48                  return value
49              value = input("Value should be between 0 and 4, input value: ")
50          except ValueError:
51              value = input("Invalid input, please enter a valid value: ")
52
53
        1 usage
54  def is_angle(value: str) -> float:
55      """Checks the value to be between 0 and 90. """
56      while True:
57          try:
58              value = float(value)
59              if 0 < value < 90:
60                  return value
61              value = input("Angle should be acute, input angle: ")
62          except ValueError:
63              value = input("Invalid input, please enter a valid value: ")
```

Task4.py

```python
def is_side(value: str) -> float:
    """Checks the value to be greater than 0. """
    while True:
        try:
            value = float(value)
            if value > 0:
                return value
            value = input("Side should be greater than 0, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")


1 usage
def menu() -> None:
    """Menu for user input. """
    print("\n1: Input the values")
    print("2: Get info about a rhombus")
    print("3: Exit\n")


1 usage
def main():
    rhombus = None
    while True:
        menu()
        command = is_command(input("Enter value: "))
        if command == 1:
            side = is_side(input("Enter the length of the side of the rhombus: "))
            angle = is_angle(input("Enter the angle of the rhombus in degrees: "))
            color = input("Enter diamond color: ")
            rhombus = Rhombus(side, angle, color)
        if command == 2:
            if rhombus is None:
                print("\nFigure not found, please try again")
                continue
            print("Area of a rhombus:", rhombus.calculate_area())
            print(rhombus.get_info())
            with open("info.txt", "w") as file:
                file.write("Area of a rhombus: {}\n".format(rhombus.calculate_area()))
                file.write("{}\n".format(rhombus.get_info()))
```

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).

а) Библиотека NumPy.

1. Создание массива. Функции array() и values().
2. Функции создания массива заданного вида.
3. Индексирование массивов NumPy. Индекс и срез.
4. Операции с массивами. Универсальные (поэлементные) функции.

б) Математические и статистические операции.

1. Функция mean()
2. Функция median()
3. Функция corrcoef()
4. Дисперсия var().
5. Стандартное отклонение std()

Вар	Условие
1.	Найти все элементы, превышающие по абсолютной величине заданное число B. Подсчитать число таких элементов и записать их в массив C. Вычислить значение медианы для этого массива C. Вычисление медианы выполнить двумя способами: через стандартную функцию и через программирование формулы.

Task5.py

```python
import numpy as np


3 usages
class Matrix:
    def __init__(self, n: int, m: int) -> None:
        self._matrix = np.random.randint(0, 100, (n, m))
        self._dimensions = (n, m)

    def __str__(self) -> str:
        return str(self._matrix)

    @property
    def dimensions(self) -> tuple:
        return self._dimensions

    2 usages
    @property
    def matrix(self) -> np.ndarray:
        return self._matrix

    @staticmethod
    def from_array(array: list) -> 'Matrix':
        """Converting an Array to a Matrix. """
        m = Matrix( n: 0,  m: 0)
        m._matrix = np.array(array)
        m._dimensions = m.matrix.shape
        return m

    def operation(self, func) -> np.ndarray:
        """Returns the result of the function. """
        return func(self._matrix)


1 usage
class UpdateMatrix(Matrix):
    def __init__(self, n: int, m: int, threshold: float) -> None:
        super().__init__(n, m)
        self._threshold = threshold
        self._filtered_elements = None
```

Task5.py

```python
    def filter_elements(self) -> np.ndarray:
        """Filters the elements of the matrix. """
        self._filtered_elements = self.matrix[np.abs(self._matrix) > self._threshold]
        return self._filtered_elements

    1 usage
    def count_filtered_elements(self) -> int:
        """Counts the number of filtered. """
        if self._filtered_elements is None:
            self.filter_elements()
        return len(self._filtered_elements)

    1 usage
    def median_of_filtered(self) -> tuple:
        """Finds the median of the filtered elements. """
        if self._filtered_elements is None:
            self.filter_elements()
        return np.median(self._filtered_elements), self.calculate_median_manually()

    1 usage
    def calculate_median_manually(self):
        """Calculates the median of the filtered elements. """
        sorted_elements = np.sort(self._filtered_elements)
        mid = len(sorted_elements) // 2
        if len(sorted_elements) % 2 == 0:
            return (sorted_elements[mid - 1] + sorted_elements[mid]) / 2
        return sorted_elements[mid]


1 usage
def is_command(value: str) -> int:
    """Checks the value to be between 0 and 3. """
    while True:
        try:
            value = int(value)
            if 0 < value < 3:
                return value
            value = input("Value should be between 0 and 3, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")
```

Task5.py

```python
def is_size(value: str) -> int:
    """ Checks the value to make sure it is greater than 0. """
    while True:
        try:
            value = int(value)
            if value > 0:
                return value
            value = input("Value should be greater than 1, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")


1 usage
def is_value(value: str) -> float:
    """ Checks the value to make sure it is greater than 0. """
    while True:
        try:
            value = float(value)
            if value > 0:
                return value
            value = input("Value should be greater than 0, input value: ")
        except ValueError:
            value = input("Invalid input, please enter a valid value: ")


1 usage
def menu() -> None:
    """Menu for user input. """
    print("\n1: Start program")
    print("2: Exit\n")
```

```python
def main():
    while True:
        menu()
        command = is_command(input("Enter value: "))
        if command == 1:
            n, m = is_size(input("Enter n: ")), is_size(input("Enter m: "))
            ths = is_value(input("Enter threshold: "))
            mat = UpdateMatrix(n, m, ths)
            print('Matrix: \n', mat)
            print(f"\nFiltered elements: {mat.filter_elements()}")
            print(f"\nCount of filtered elements: {mat.count_filtered_elements()}")
            print(f"\nMedians: {mat.median_of_filtered()}")
        if command == 2:
            break
```