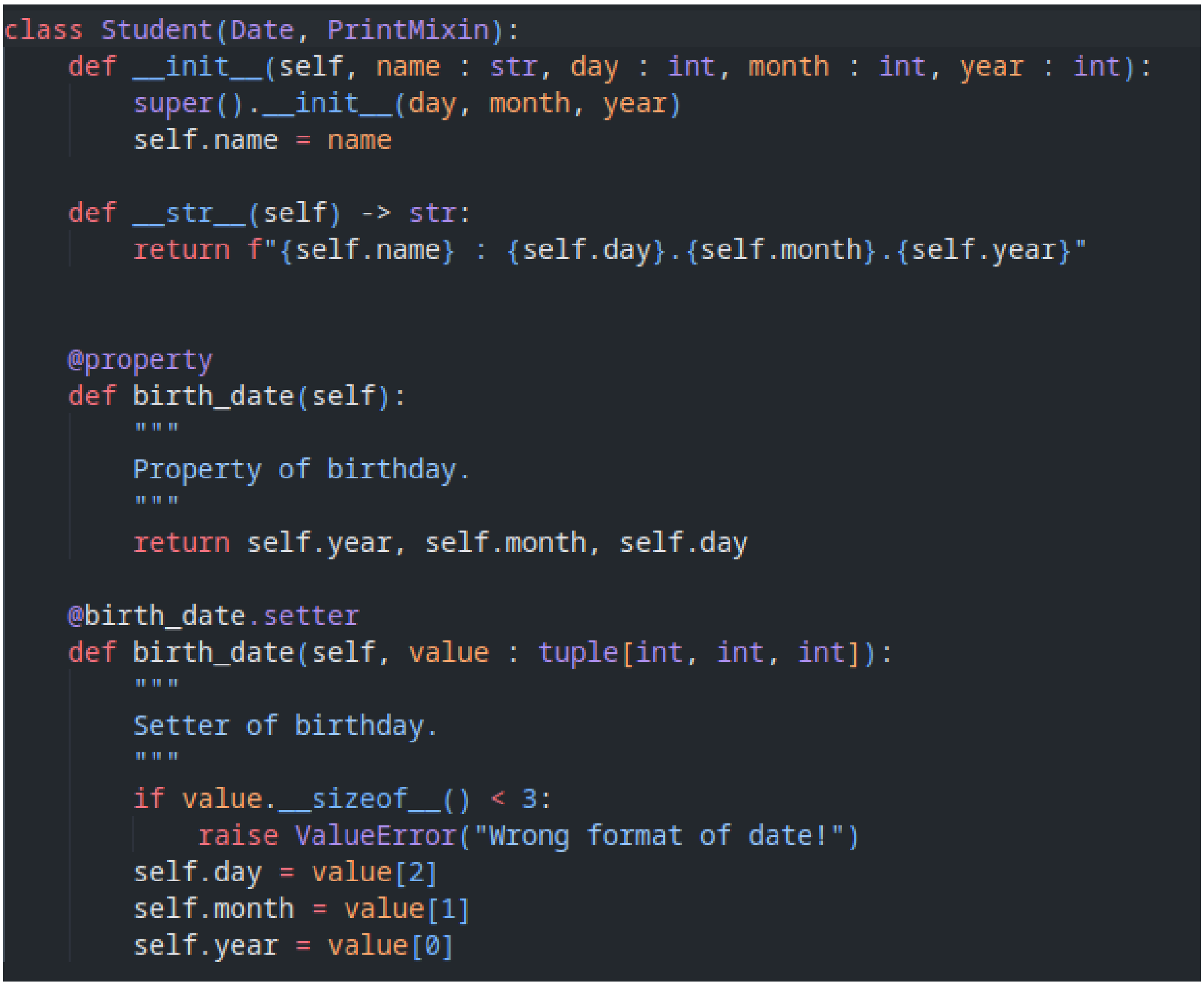
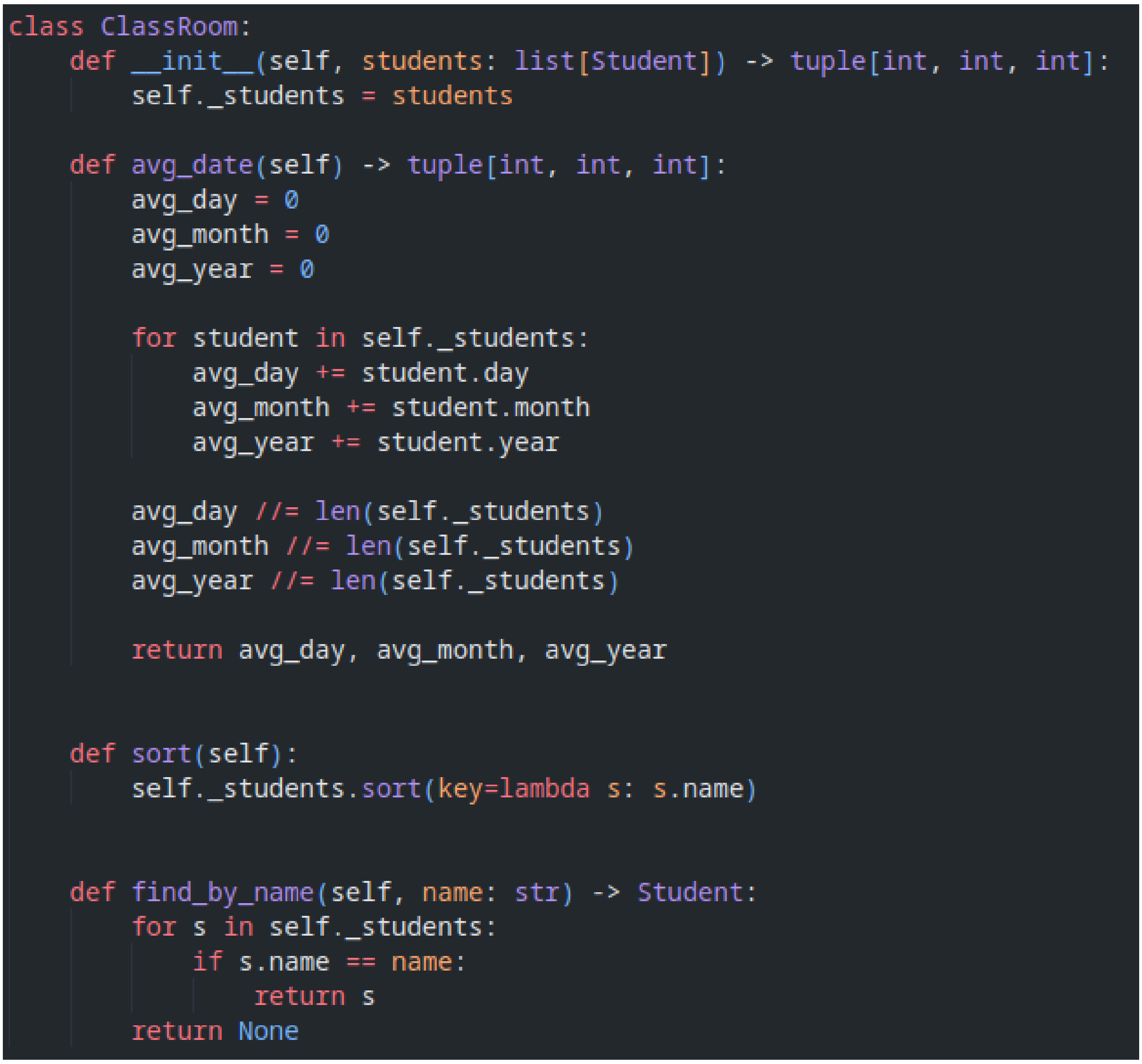
Лабораторная работа №4

Сергеев. Р.В. 253502

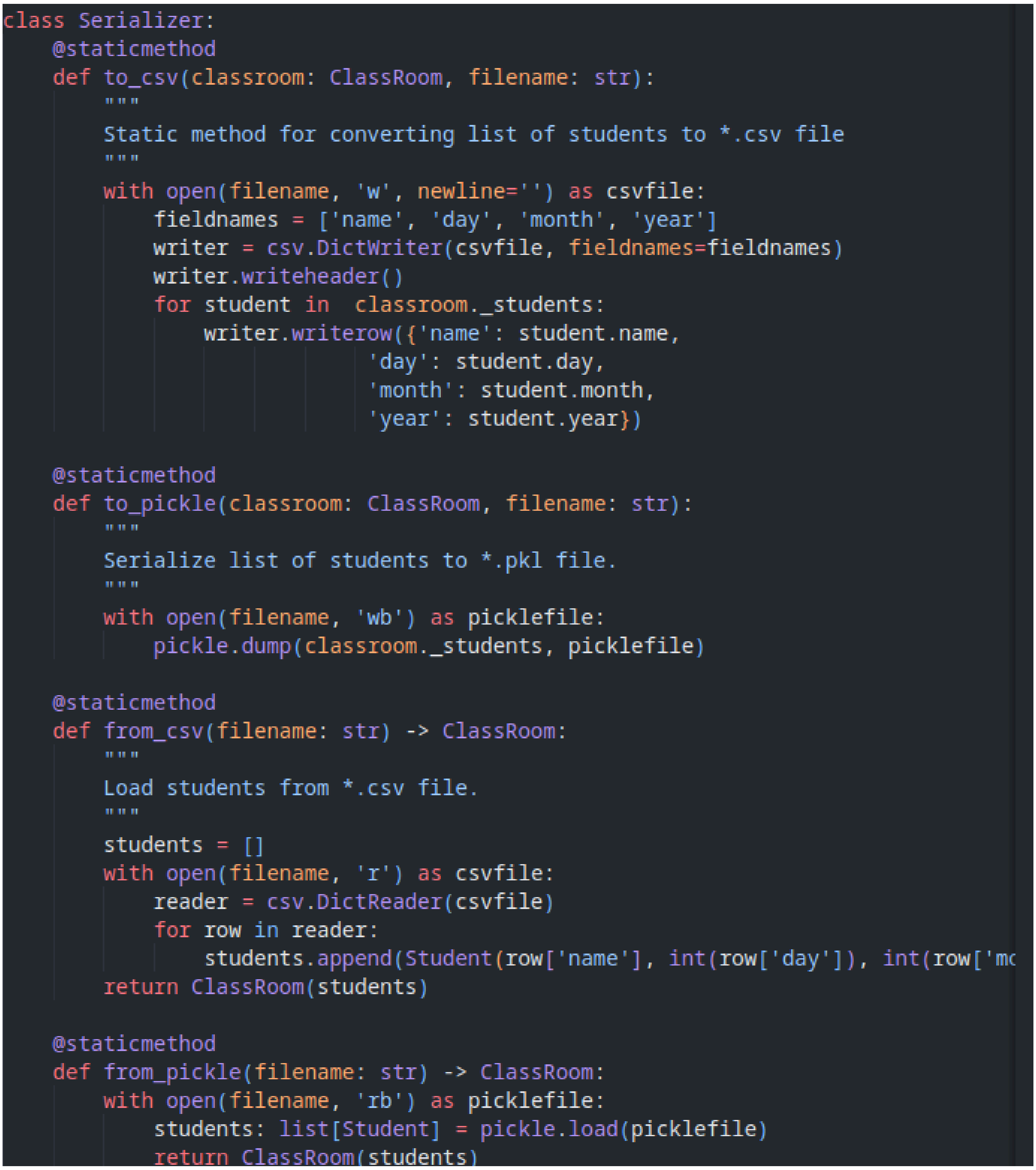
Задание 1



```python
class Student(Date, PrintMixin):
    def __init__(self, name : str, day : int, month : int, year : int):
        super().__init__(day, month, year)
        self.name = name

    def __str__(self) -> str:
        return f"{self.name} : {self.day}.{self.month}.{self.year}"


    @property
    def birth_date(self):
        """
        Property of birthday.
        """
        return self.year, self.month, self.day

    @birth_date.setter
    def birth_date(self, value : tuple[int, int, int]):
        """
        Setter of birthday.
        """
        if value.__sizeof__() < 3:
            raise ValueError("Wrong format of date!")
        self.day = value[2]
        self.month = value[1]
        self.year = value[0]
```

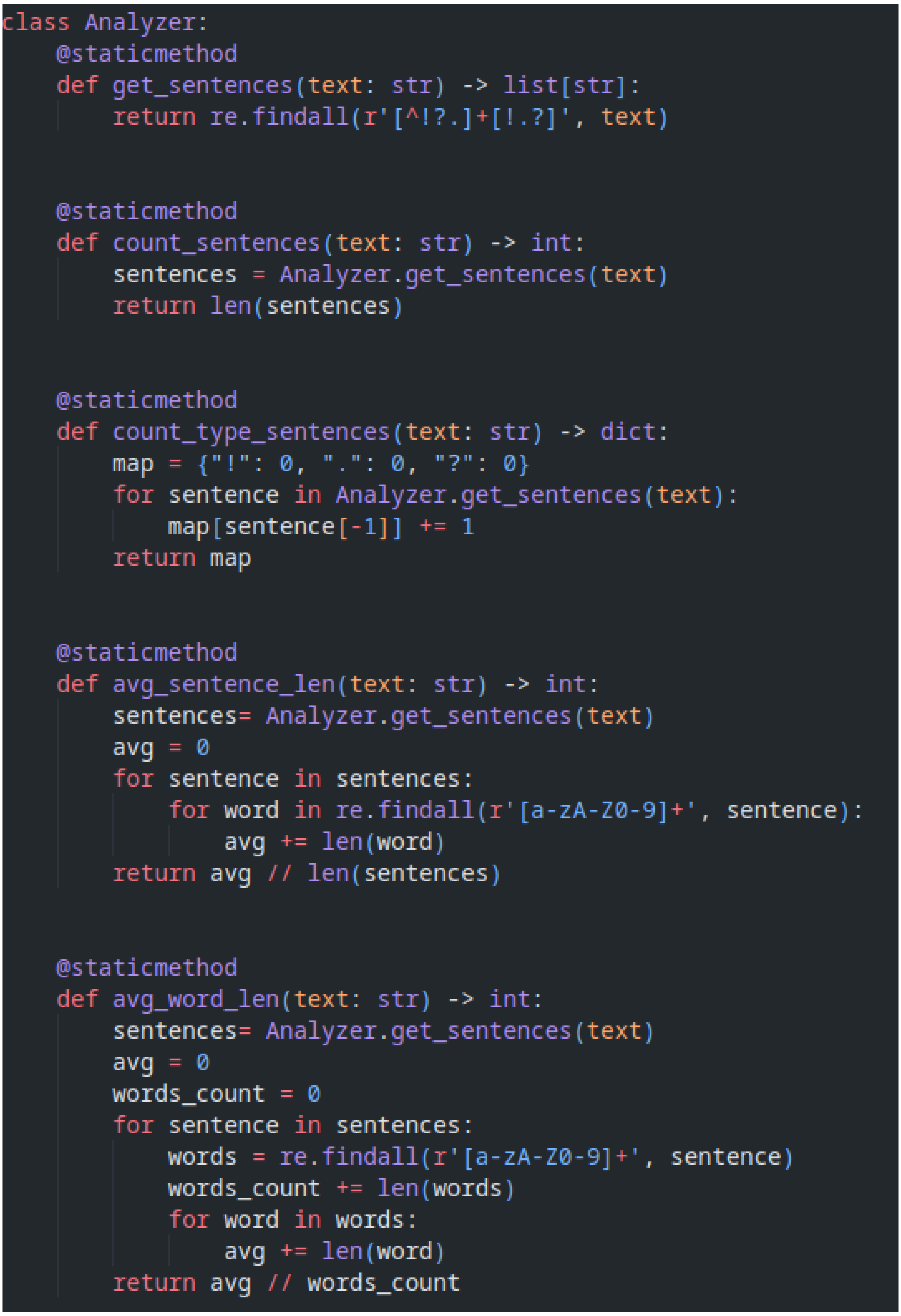


```python
class ClassRoom:
    def __init__(self, students: list[Student]) -> tuple[int, int, int]:
        self._students = students

    def avg_date(self) -> tuple[int, int, int]:
        avg_day = 0
        avg_month = 0
        avg_year = 0

        for student in self._students:
            avg_day += student.day
            avg_month += student.month
            avg_year += student.year

        avg_day //= len(self._students)
        avg_month //= len(self._students)
        avg_year //= len(self._students)

        return avg_day, avg_month, avg_year


    def sort(self):
        self._students.sort(key=lambda s: s.name)


    def find_by_name(self, name: str) -> Student:
        for s in self._students:
            if s.name == name:
                return s
        return None
```

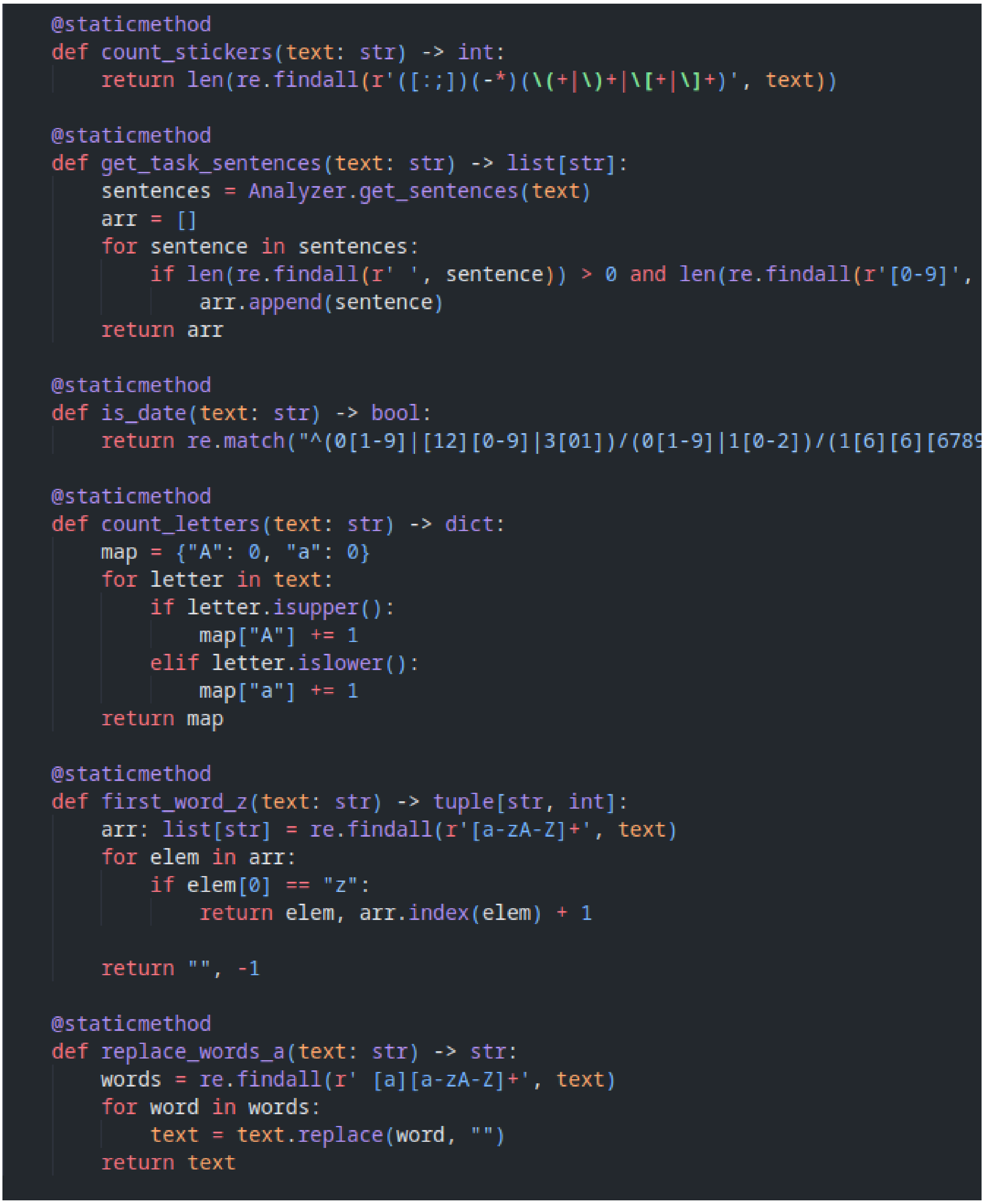


```
class Serializer:
    @staticmethod
    def to_csv(classroom: ClassRoom, filename: str):
        """
        Static method for converting list of students to *.csv file
        """
        with open(filename, 'w', newline='') as csvfile:
            fieldnames = ['name', 'day', 'month', 'year']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for student in  classroom._students:
                writer.writerow({'name': student.name,
                             'day': student.day,
                             'month': student.month,
                             'year': student.year})

    @staticmethod
    def to_pickle(classroom: ClassRoom, filename: str):
        """
        Serialize list of students to *.pkl file.
        """
        with open(filename, 'wb') as picklefile:
            pickle.dump(classroom._students, picklefile)

    @staticmethod
    def from_csv(filename: str) -> ClassRoom:
        """
        Load students from *.csv file.
        """
        students = []
        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                students.append(Student(row['name'], int(row['day']), int(row['mo
        return ClassRoom(students)

    @staticmethod
    def from_pickle(filename: str) -> ClassRoom:
        with open(filename, 'rb') as picklefile:
            students: list[Student] = pickle.load(picklefile)
            return ClassRoom(students)
```

Задание 2

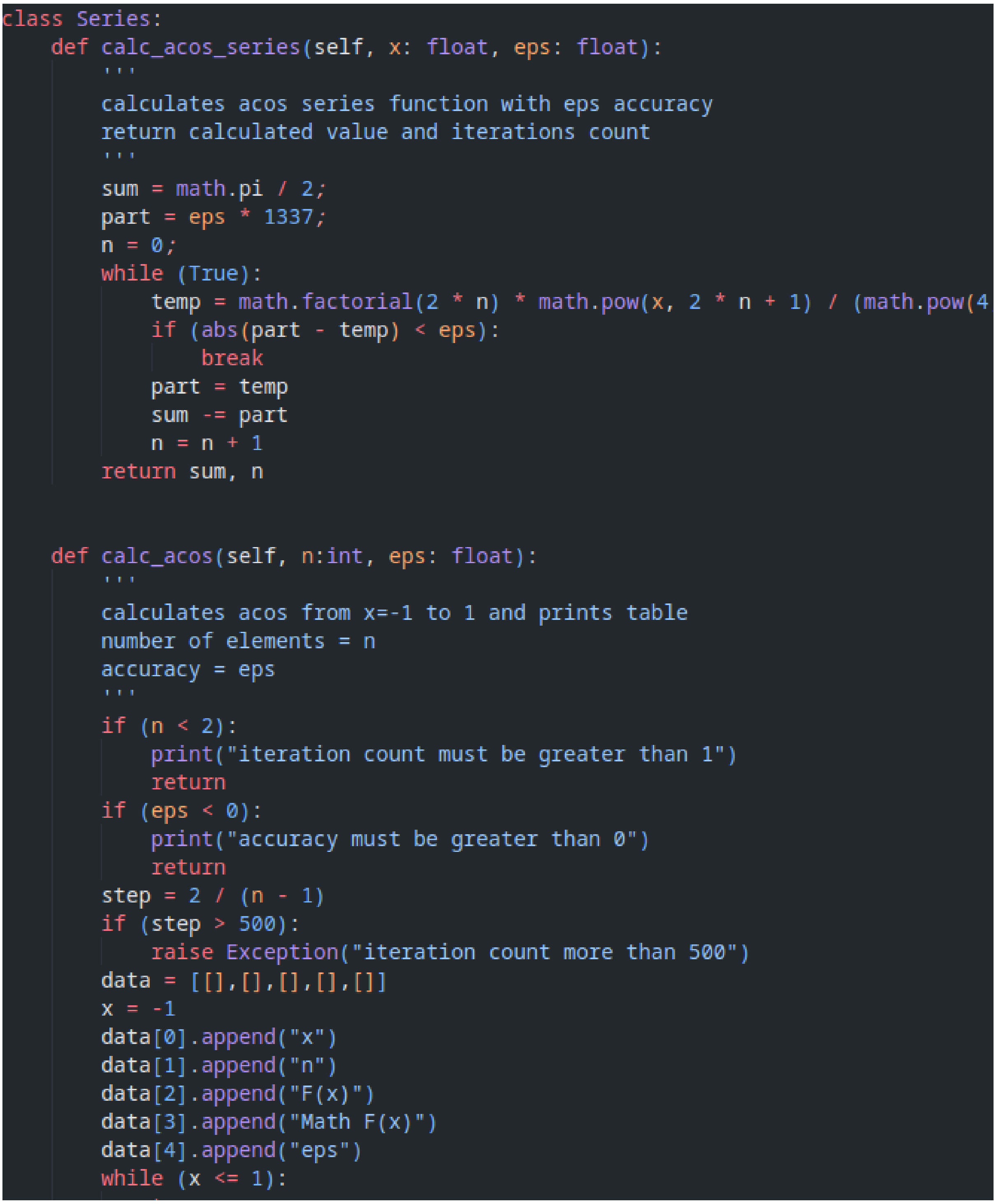


```python
class Analyzer:
    @staticmethod
    def get_sentences(text: str) -> list[str]:
        return re.findall(r'[^!?.]+[!.?]', text)


    @staticmethod
    def count_sentences(text: str) -> int:
        sentences = Analyzer.get_sentences(text)
        return len(sentences)


    @staticmethod
    def count_type_sentences(text: str) -> dict:
        map = {"!": 0, ".": 0, "?": 0}
        for sentence in Analyzer.get_sentences(text):
            map[sentence[-1]] += 1
        return map


    @staticmethod
    def avg_sentence_len(text: str) -> int:
        sentences= Analyzer.get_sentences(text)
        avg = 0
        for sentence in sentences:
            for word in re.findall(r'[a-zA-Z0-9]+', sentence):
                avg += len(word)
        return avg // len(sentences)


    @staticmethod
    def avg_word_len(text: str) -> int:
        sentences= Analyzer.get_sentences(text)
        avg = 0
        words_count = 0
        for sentence in sentences:
            words = re.findall(r'[a-zA-Z0-9]+', sentence)
            words_count += len(words)
            for word in words:
                avg += len(word)
        return avg // words_count
```

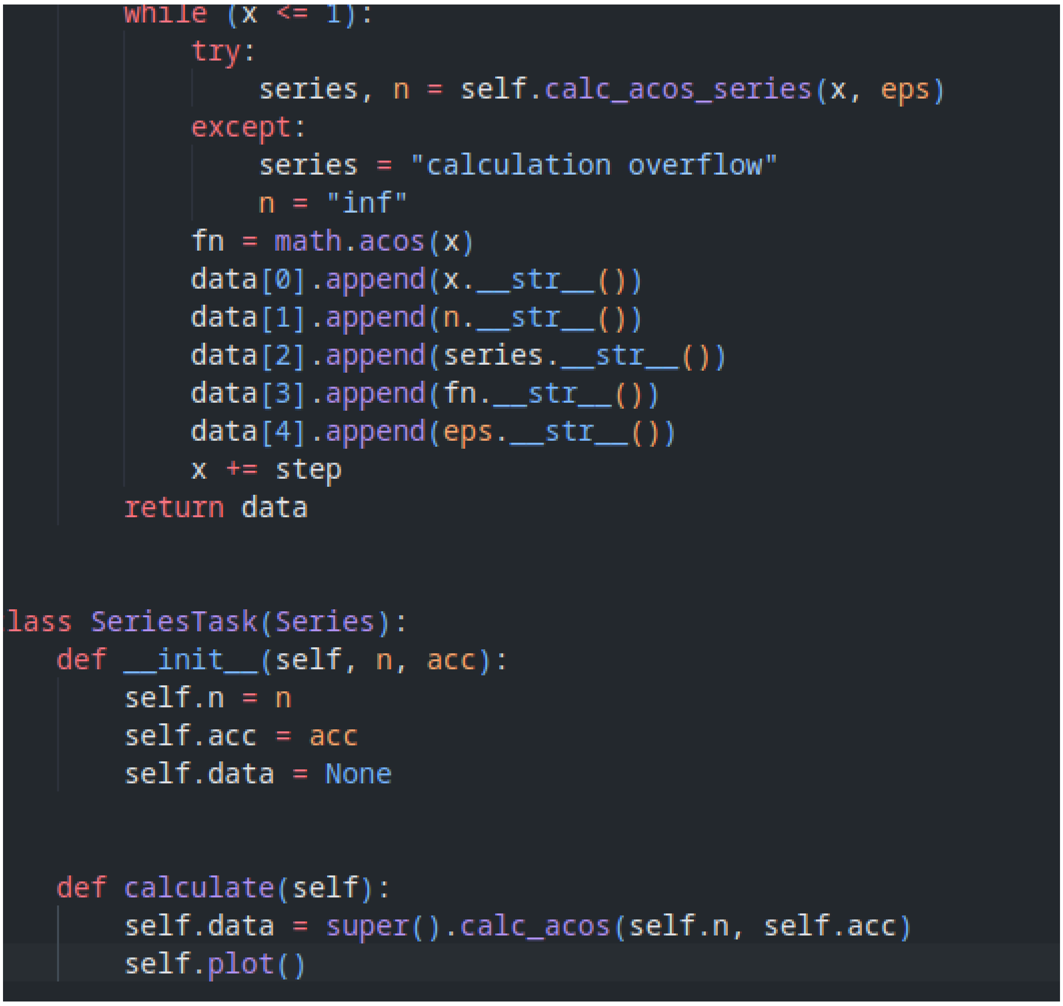


```python
    @staticmethod
    def count_stickers(text: str) -> int:
        return len(re.findall(r'([:;])(-*)(\(+|\)+|\[+|\]+)', text))

    @staticmethod
    def get_task_sentences(text: str) -> list[str]:
        sentences = Analyzer.get_sentences(text)
        arr = []
        for sentence in sentences:
            if len(re.findall(r' ', sentence)) > 0 and len(re.findall(r'[0-9]',
                arr.append(sentence)
        return arr

    @staticmethod
    def is_date(text: str) -> bool:
        return re.match("^(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[0-2])/(1[6][6][6789

    @staticmethod
    def count_letters(text: str) -> dict:
        map = {"A": 0, "a": 0}
        for letter in text:
            if letter.isupper():
                map["A"] += 1
            elif letter.islower():
                map["a"] += 1
        return map

    @staticmethod
    def first_word_z(text: str) -> tuple[str, int]:
        arr: list[str] = re.findall(r'[a-zA-Z]+', text)
        for elem in arr:
            if elem[0] == "z":
                return elem, arr.index(elem) + 1

        return "", -1

    @staticmethod
    def replace_words_a(text: str) -> str:
        words = re.findall(r' [a][a-zA-Z]+', text)
        for word in words:
            text = text.replace(word, "")
        return text
```

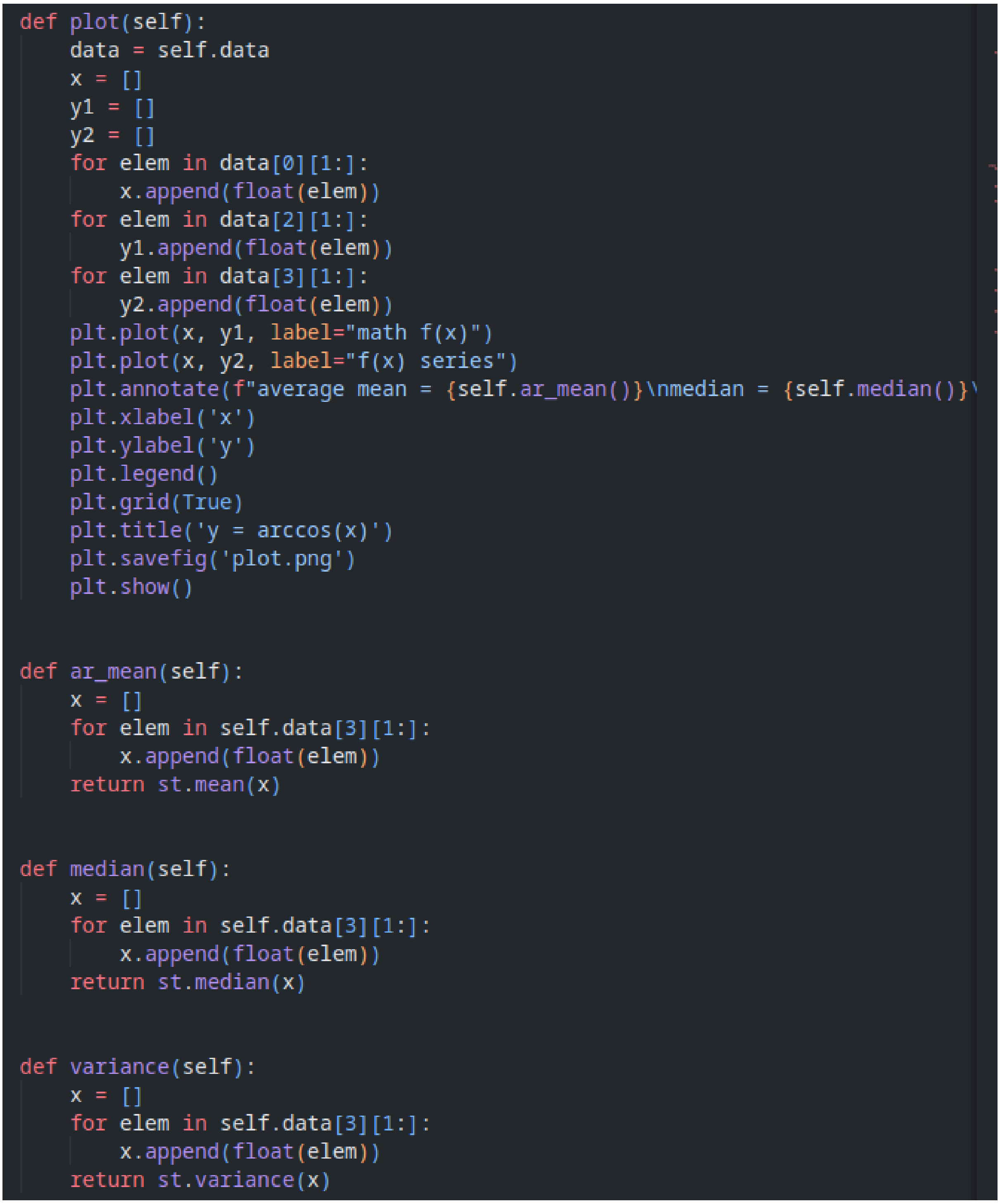
Задание 3



```python
class Series:
    def calc_acos_series(self, x: float, eps: float):
        '''
        calculates acos series function with eps accuracy
        return calculated value and iterations count
        '''
        sum = math.pi / 2;
        part = eps * 1337;
        n = 0;
        while (True):
            temp = math.factorial(2 * n) * math.pow(x, 2 * n + 1) / (math.pow(4
            if (abs(part - temp) < eps):
                break
            part = temp
            sum -= part
            n = n + 1
        return sum, n


    def calc_acos(self, n:int, eps: float):
        '''
        calculates acos from x=-1 to 1 and prints table
        number of elements = n
        accuracy = eps
        '''
        if (n < 2):
            print("iteration count must be greater than 1")
            return
        if (eps < 0):
            print("accuracy must be greater than 0")
            return
        step = 2 / (n - 1)
        if (step > 500):
            raise Exception("iteration count more than 500")
        data = [[],[],[],[],[]]
        x = -1
        data[0].append("x")
        data[1].append("n")
        data[2].append("F(x)")
        data[3].append("Math F(x)")
        data[4].append("eps")
        while (x <= 1):
```

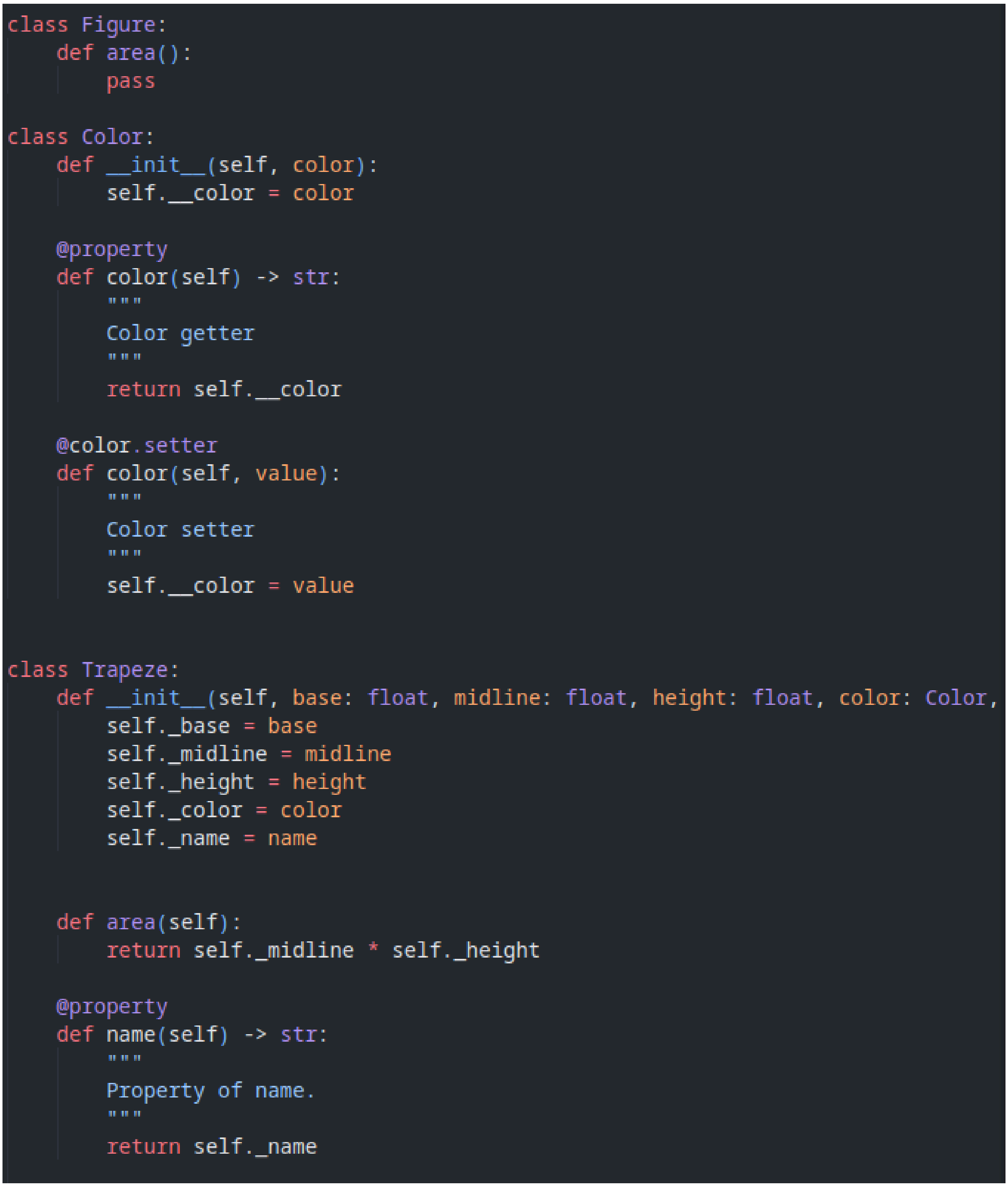


```
        while (x <= 1):
            try:
                series, n = self.calc_acos_series(x, eps)
            except:
                series = "calculation overflow"
                n = "inf"
            fn = math.acos(x)
            data[0].append(x.__str__())
            data[1].append(n.__str__())
            data[2].append(series.__str__())
            data[3].append(fn.__str__())
            data[4].append(eps.__str__())
            x += step
        return data


lass SeriesTask(Series):
    def __init__(self, n, acc):
        self.n = n
        self.acc = acc
        self.data = None


    def calculate(self):
        self.data = super().calc_acos(self.n, self.acc)
        self.plot()
```

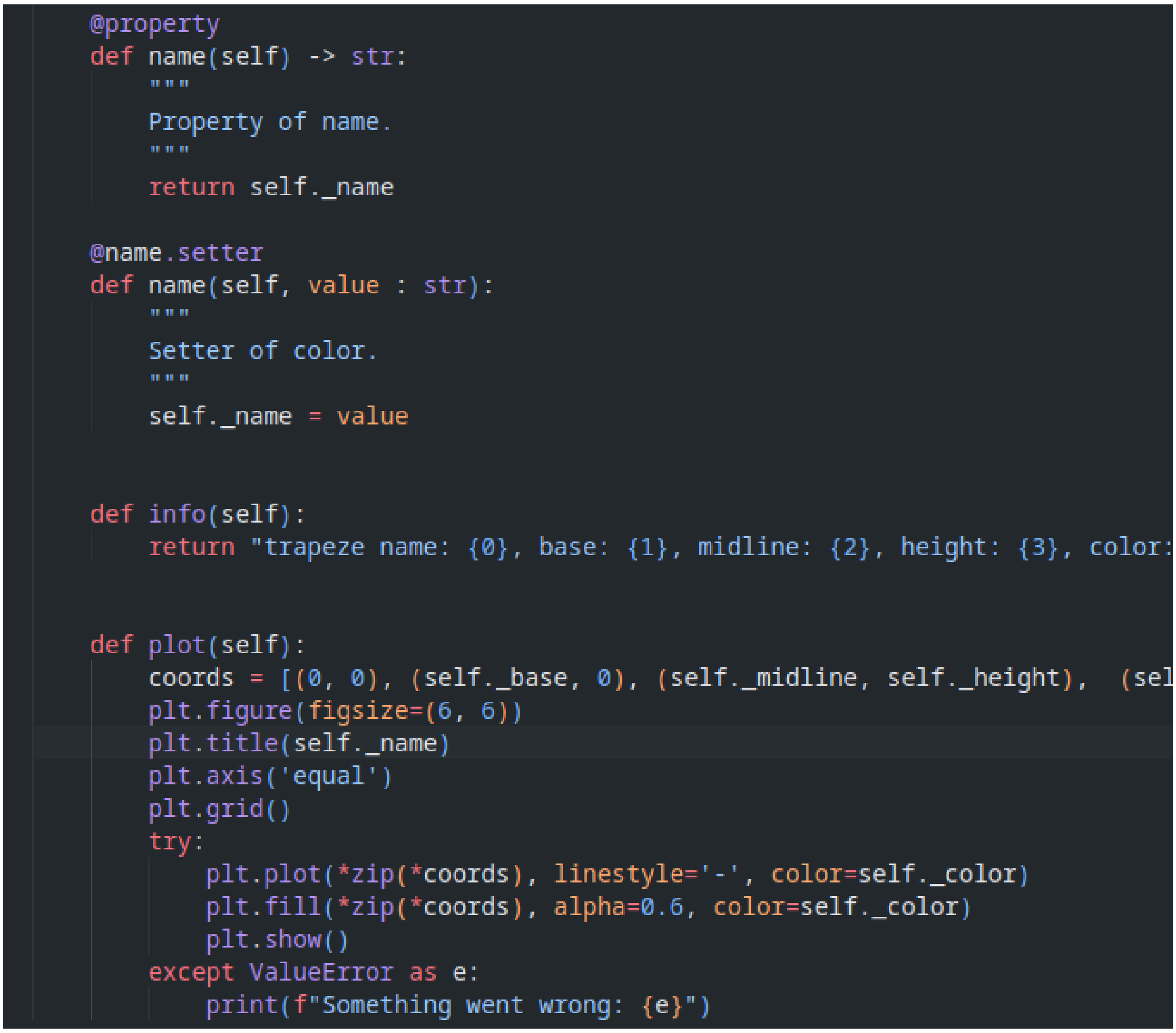


```
def plot(self):
    data = self.data
    x = []
    y1 = []
    y2 = []
    for elem in data[0][1:]:
        x.append(float(elem))
    for elem in data[2][1:]:
        y1.append(float(elem))
    for elem in data[3][1:]:
        y2.append(float(elem))
    plt.plot(x, y1, label="math f(x)")
    plt.plot(x, y2, label="f(x) series")
    plt.annotate(f"average mean = {self.ar_mean()}\nmedian = {self.median()}\
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)
    plt.title('y = arccos(x)')
    plt.savefig('plot.png')
    plt.show()


def ar_mean(self):
    x = []
    for elem in self.data[3][1:]:
        x.append(float(elem))
    return st.mean(x)


def median(self):
    x = []
    for elem in self.data[3][1:]:
        x.append(float(elem))
    return st.median(x)


def variance(self):
    x = []
    for elem in self.data[3][1:]:
        x.append(float(elem))
    return st.variance(x)
```

Задание 4

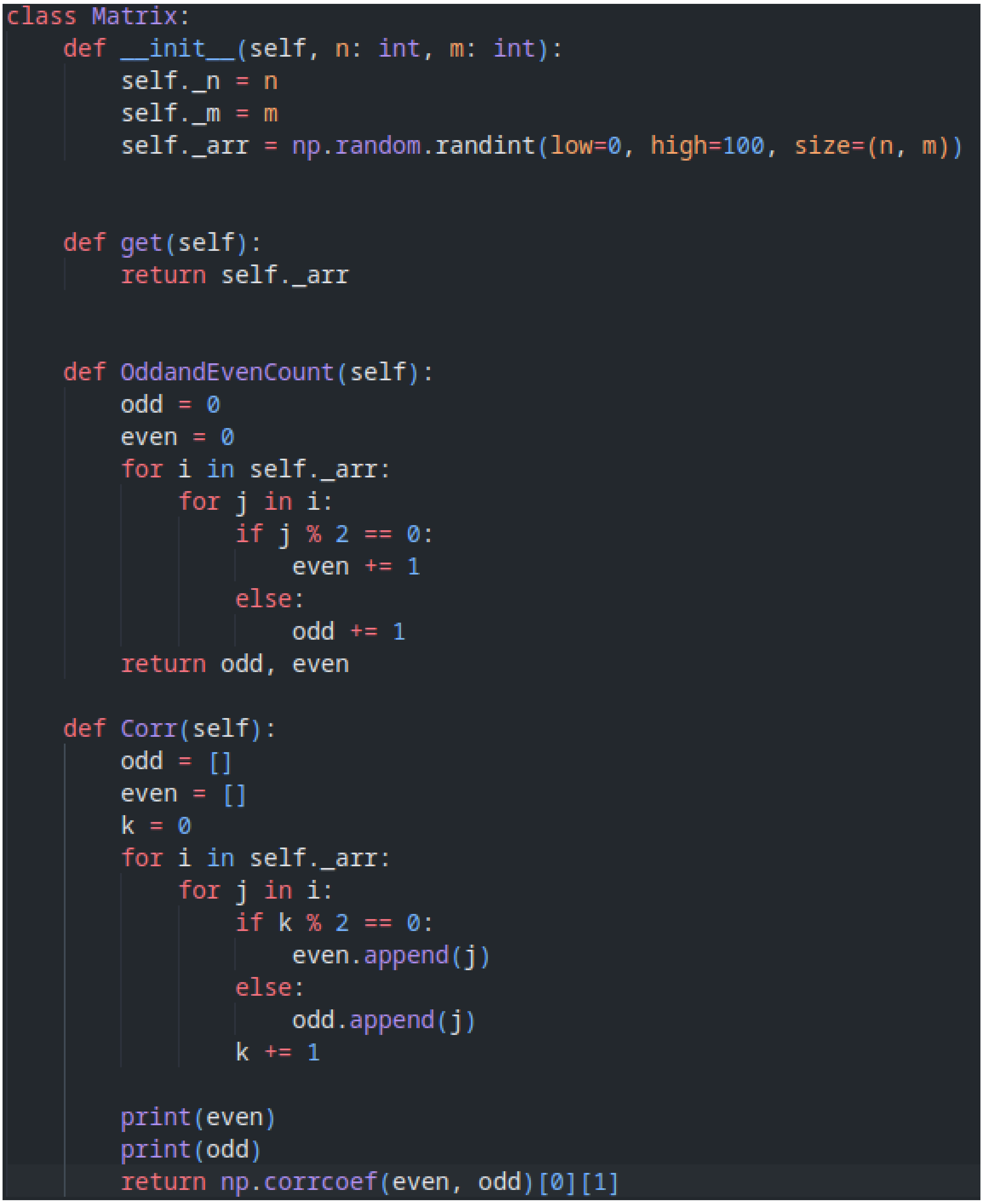


```python
class Figure:
    def area():
        pass

class Color:
    def __init__(self, color):
        self.__color = color

    @property
    def color(self) -> str:
        """
        Color getter
        """
        return self.__color

    @color.setter
    def color(self, value):
        """
        Color setter
        """
        self.__color = value


class Trapeze:
    def __init__(self, base: float, midline: float, height: float, color: Color,
        self._base = base
        self._midline = midline
        self._height = height
        self._color = color
        self._name = name


    def area(self):
        return self._midline * self._height

    @property
    def name(self) -> str:
        """
        Property of name.
        """
        return self._name
```



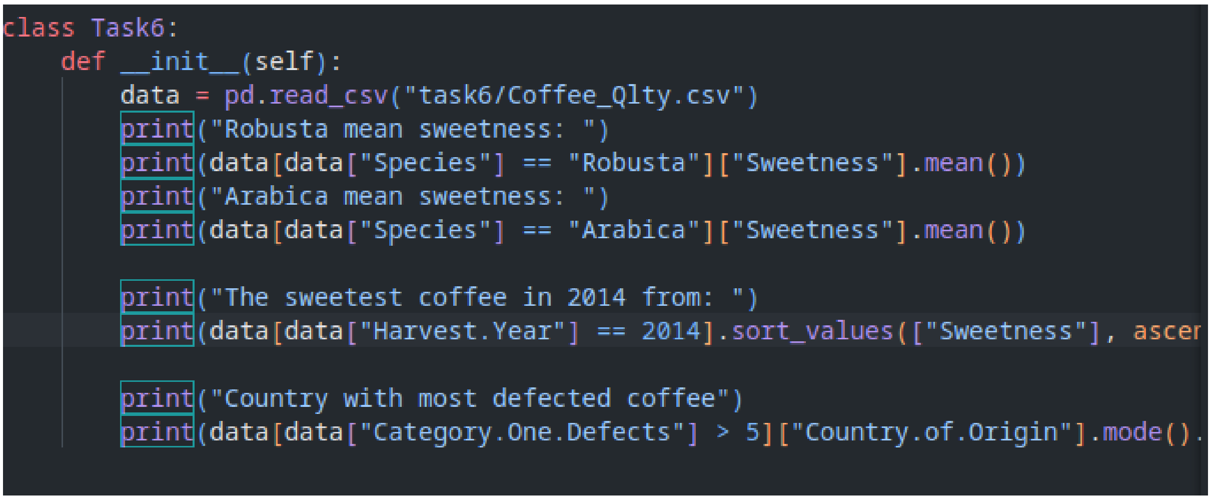
```
    @property
    def name(self) -> str:
        """
        Property of name.
        """
        return self._name

    @name.setter
    def name(self, value : str):
        """
        Setter of color.
        """
        self._name = value


    def info(self):
        return "trapeze name: {0}, base: {1}, midline: {2}, height: {3}, color:


    def plot(self):
        coords = [(0, 0), (self._base, 0), (self._midline, self._height),  (sel
        plt.figure(figsize=(6, 6))
        plt.title(self._name)
        plt.axis('equal')
        plt.grid()
        try:
            plt.plot(*zip(*coords), linestyle='-', color=self._color)
            plt.fill(*zip(*coords), alpha=0.6, color=self._color)
            plt.show()
        except ValueError as e:
            print(f"Something went wrong: {e}")
```

Задание 5



```
class Matrix:
    def __init__(self, n: int, m: int):
        self._n = n
        self._m = m
        self._arr = np.random.randint(low=0, high=100, size=(n, m))


    def get(self):
        return self._arr


    def OddandEvenCount(self):
        odd = 0
        even = 0
        for i in self._arr:
            for j in i:
                if j % 2 == 0:
                    even += 1
                else:
                    odd += 1
        return odd, even

    def Corr(self):
        odd = []
        even = []
        k = 0
        for i in self._arr:
            for j in i:
                if k % 2 == 0:
                    even.append(j)
                else:
                    odd.append(j)
                k += 1

        print(even)
        print(odd)
        return np.corrcoef(even, odd)[0][1]
```

Задание 6



```
class Task6:
    def __init__(self):
        data = pd.read_csv("task6/Coffee_Qlty.csv")
        print("Robusta mean sweetness: ")
        print(data[data["Species"] == "Robusta"]["Sweetness"].mean())
        print("Arabica mean sweetness: ")
        print(data[data["Species"] == "Arabica"]["Sweetness"].mean())

        print("The sweetest coffee in 2014 from: ")
        print(data[data["Harvest.Year"] == 2014].sort_values(["Sweetness"], ascer

        print("Country with most defected coffee")
        print(data[data["Category.One.Defects"] > 5]["Country.of.Origin"].mode().
```