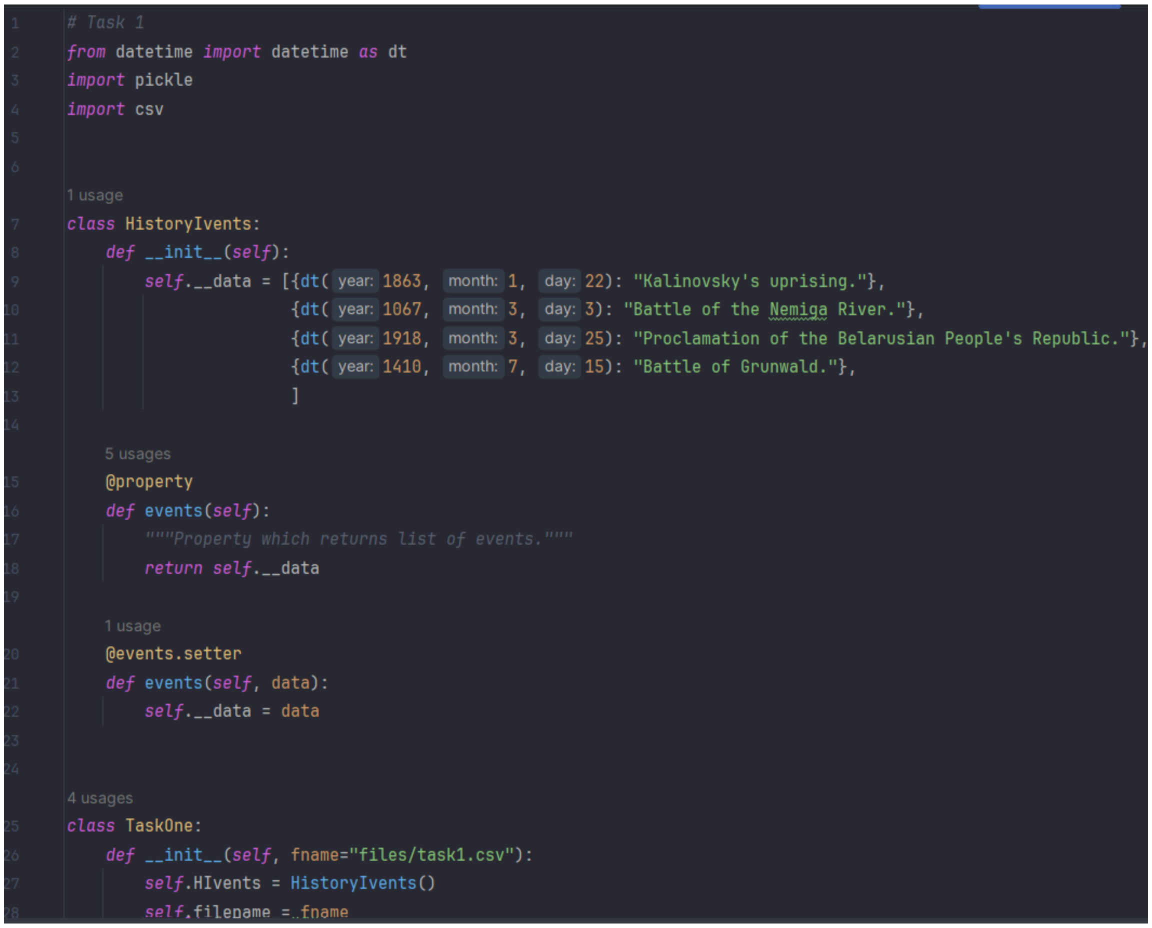
Лабораторная работа №4

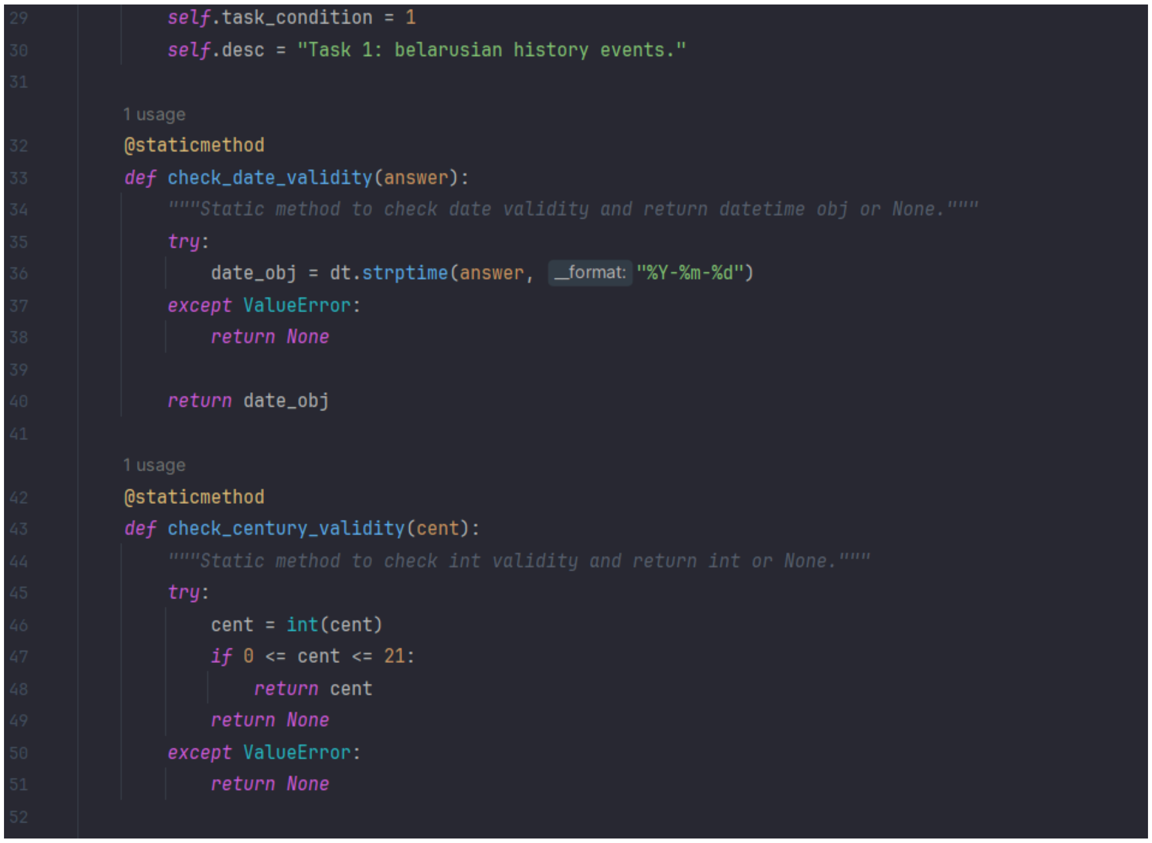
Тема: Работа с файлами, классами, сериализаторами, регулярными выражениями и стандартными библиотеками.

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: Барановский Г.В. Гр.253502

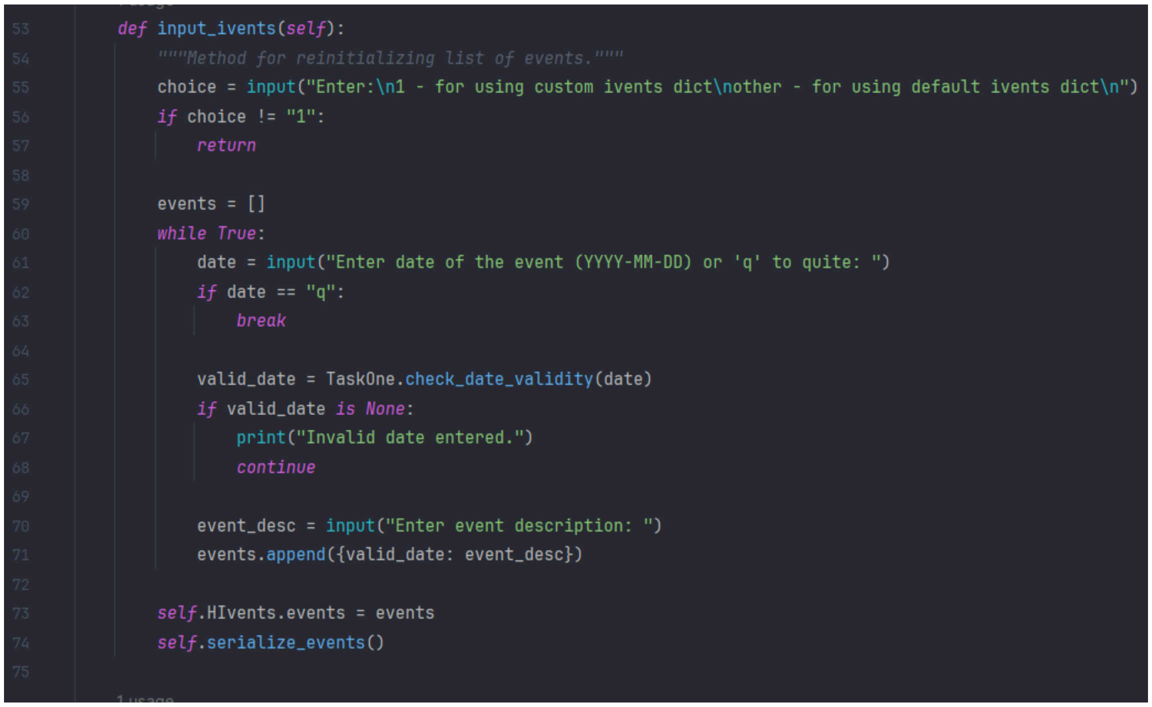
1. Задание 1:



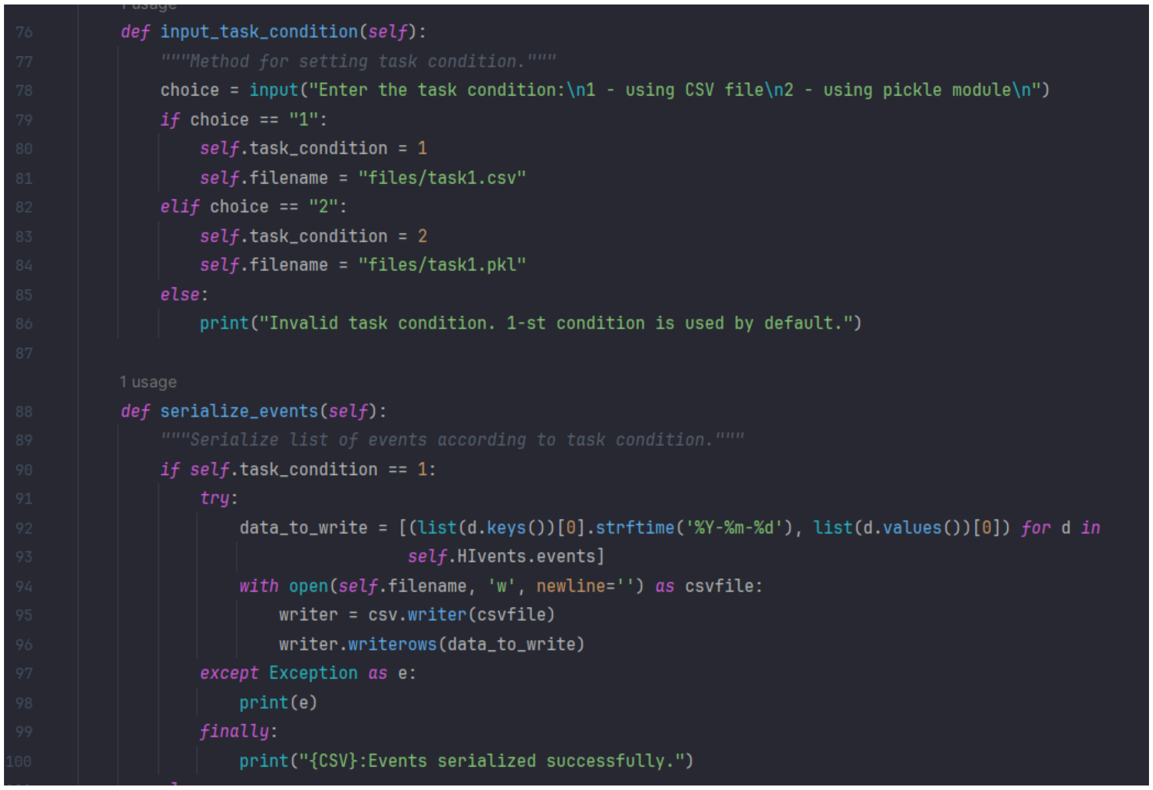
```
# Task 1
from datetime import datetime as dt
import pickle
import csv


1 usage
class HistoryIvents:
    def __init__(self):
        self.__data = [{dt( year: 1863,  month: 1,  day: 22): "Kalinovsky's uprising."},
                       {dt( year: 1067,  month: 3,  day: 3): "Battle of the Nemiga River."},
                       {dt( year: 1918,  month: 3,  day: 25): "Proclamation of the Belarusian People's Republic."},
                       {dt( year: 1410,  month: 7,  day: 15): "Battle of Grunwald."},
                       ]

    5 usages
    @property
    def events(self):
        """Property which returns list of events."""
        return self.__data

    1 usage
    @events.setter
    def events(self, data):
        self.__data = data


4 usages
class TaskOne:
    def __init__(self, fname="files/task1.csv"):
        self.HIvents = HistoryIvents()
        self.filename = fname
```



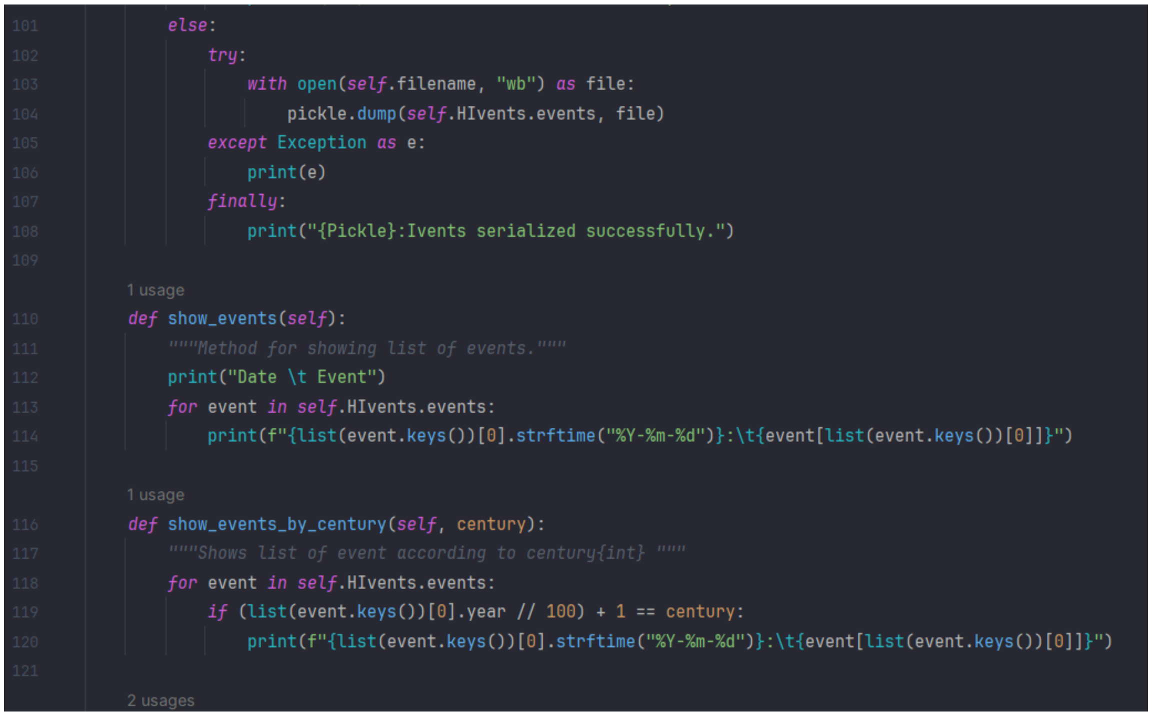
```python
        self.task_condition = 1
        self.desc = "Task 1: belarusian history events."

    1 usage
    @staticmethod
    def check_date_validity(answer):
        """Static method to check date validity and return datetime obj or None."""
        try:
            date_obj = dt.strptime(answer, _format: "%Y-%m-%d")
        except ValueError:
            return None

        return date_obj

    1 usage
    @staticmethod
    def check_century_validity(cent):
        """Static method to check int validity and return int or None."""
        try:
            cent = int(cent)
            if 0 <= cent <= 21:
                return cent
            return None
        except ValueError:
            return None

```



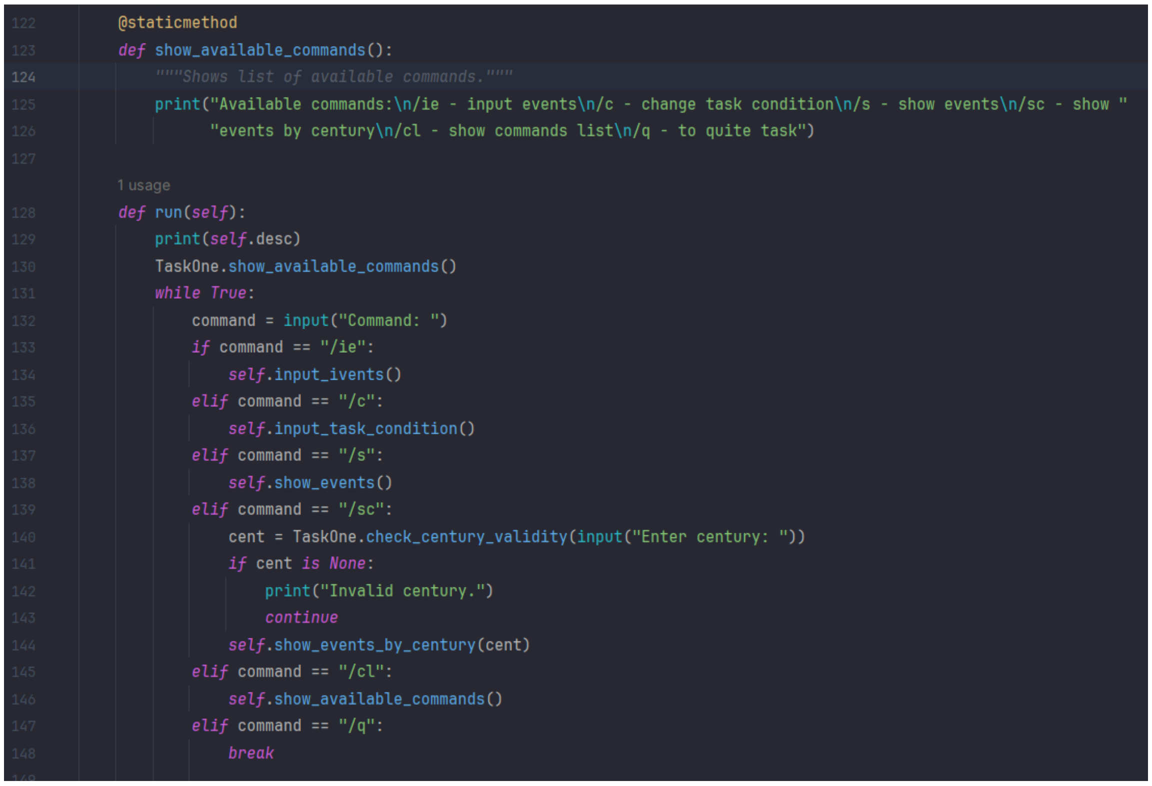
```python
    def input_ivents(self):
        """Method for reinitializing list of events."""
        choice = input("Enter:\n1 - for using custom ivents dict\nother - for using default ivents dict\n")
        if choice != "1":
            return

        events = []
        while True:
            date = input("Enter date of the event (YYYY-MM-DD) or 'q' to quite: ")
            if date == "q":
                break

            valid_date = TaskOne.check_date_validity(date)
            if valid_date is None:
                print("Invalid date entered.")
                continue

            event_desc = input("Enter event description: ")
            events.append({valid_date: event_desc})

        self.HIvents.events = events
        self.serialize_events()

```



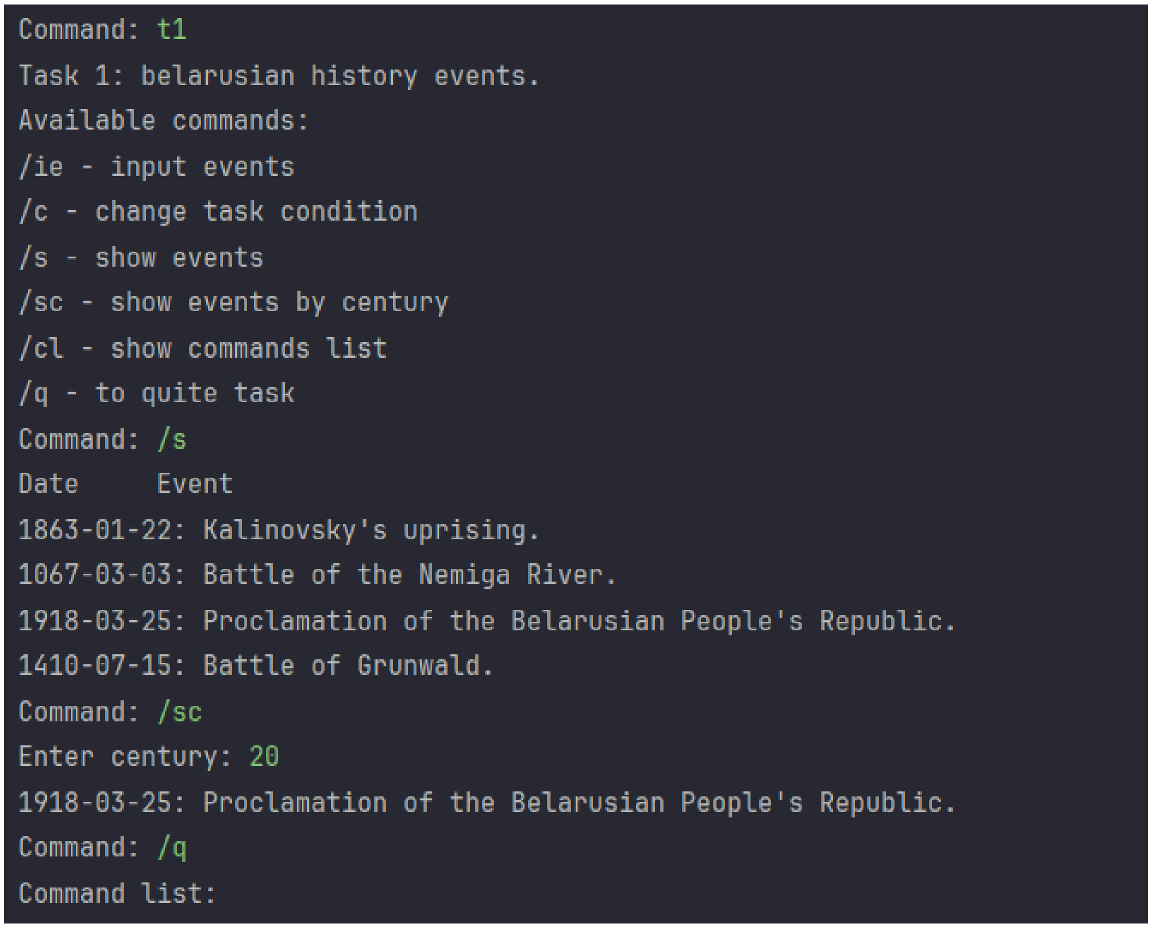
```python
    def input_task_condition(self):
        """Method for setting task condition."""
        choice = input("Enter the task condition:\n1 - using CSV file\n2 - using pickle module\n")
        if choice == "1":
            self.task_condition = 1
            self.filename = "files/task1.csv"
        elif choice == "2":
            self.task_condition = 2
            self.filename = "files/task1.pkl"
        else:
            print("Invalid task condition. 1-st condition is used by default.")

    1 usage
    def serialize_events(self):
        """Serialize list of events according to task condition."""
        if self.task_condition == 1:
            try:
                data_to_write = [(list(d.keys())[0].strftime('%Y-%m-%d'), list(d.values())[0]) for d in
                                 self.HIvents.events]
                with open(self.filename, 'w', newline='') as csvfile:
                    writer = csv.writer(csvfile)
                    writer.writerows(data_to_write)
            except Exception as e:
                print(e)
            finally:
                print("{CSV}:Events serialized successfully.")
```



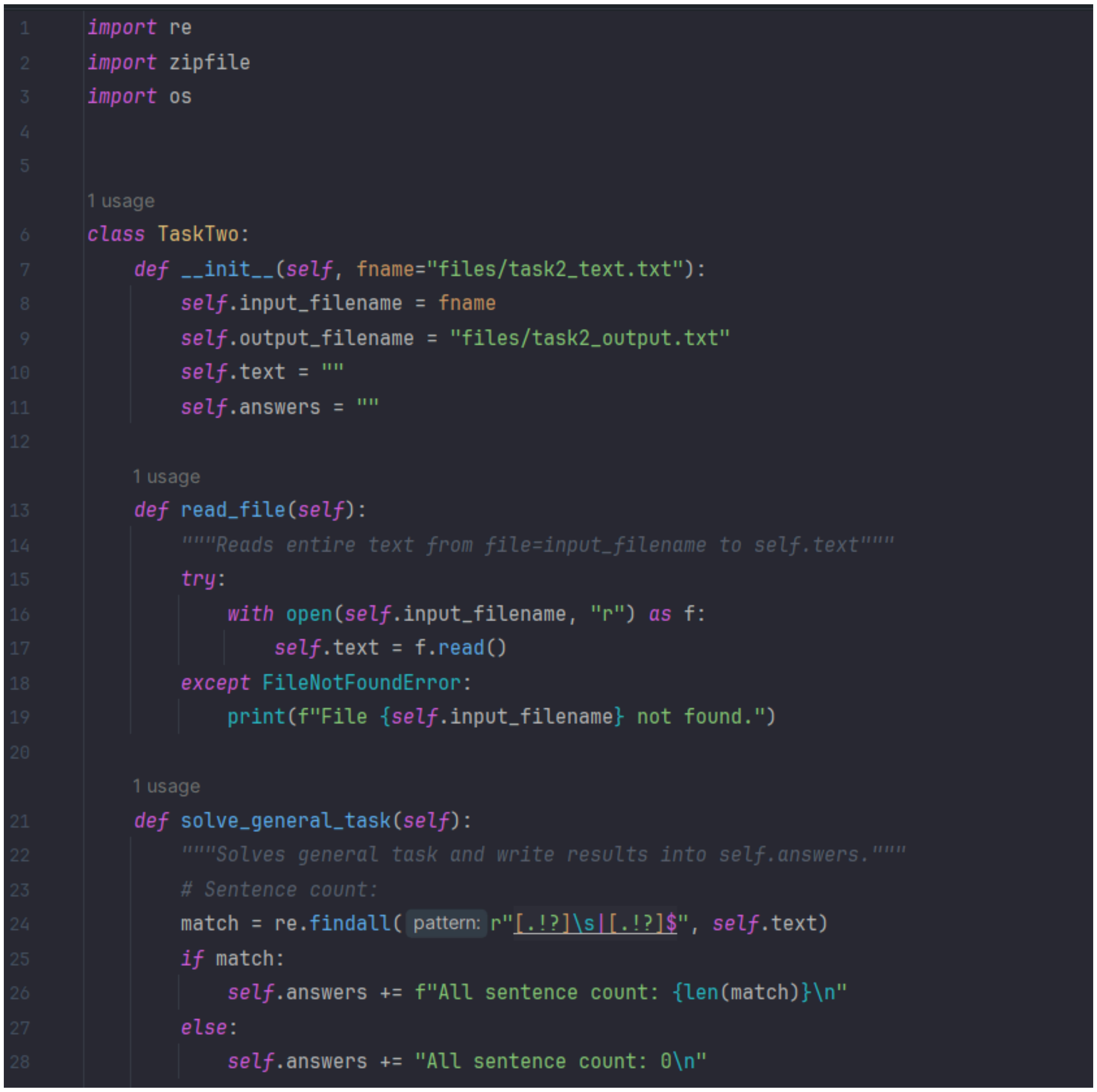
```python
        else:
            try:
                with open(self.filename, "wb") as file:
                    pickle.dump(self.HIvents.events, file)
            except Exception as e:
                print(e)
            finally:
                print("{Pickle}:Ivents serialized successfully.")

    1 usage
    def show_events(self):
        """Method for showing list of events."""
        print("Date \t Event")
        for event in self.HIvents.events:
            print(f"{list(event.keys())[0].strftime("%Y-%m-%d")}:\t{event[list(event.keys())[0]]}")

    1 usage
    def show_events_by_century(self, century):
        """Shows list of event according to century{int} """
        for event in self.HIvents.events:
            if (list(event.keys())[0].year // 100) + 1 == century:
                print(f"{list(event.keys())[0].strftime("%Y-%m-%d")}:\t{event[list(event.keys())[0]]}")

    2 usages
```



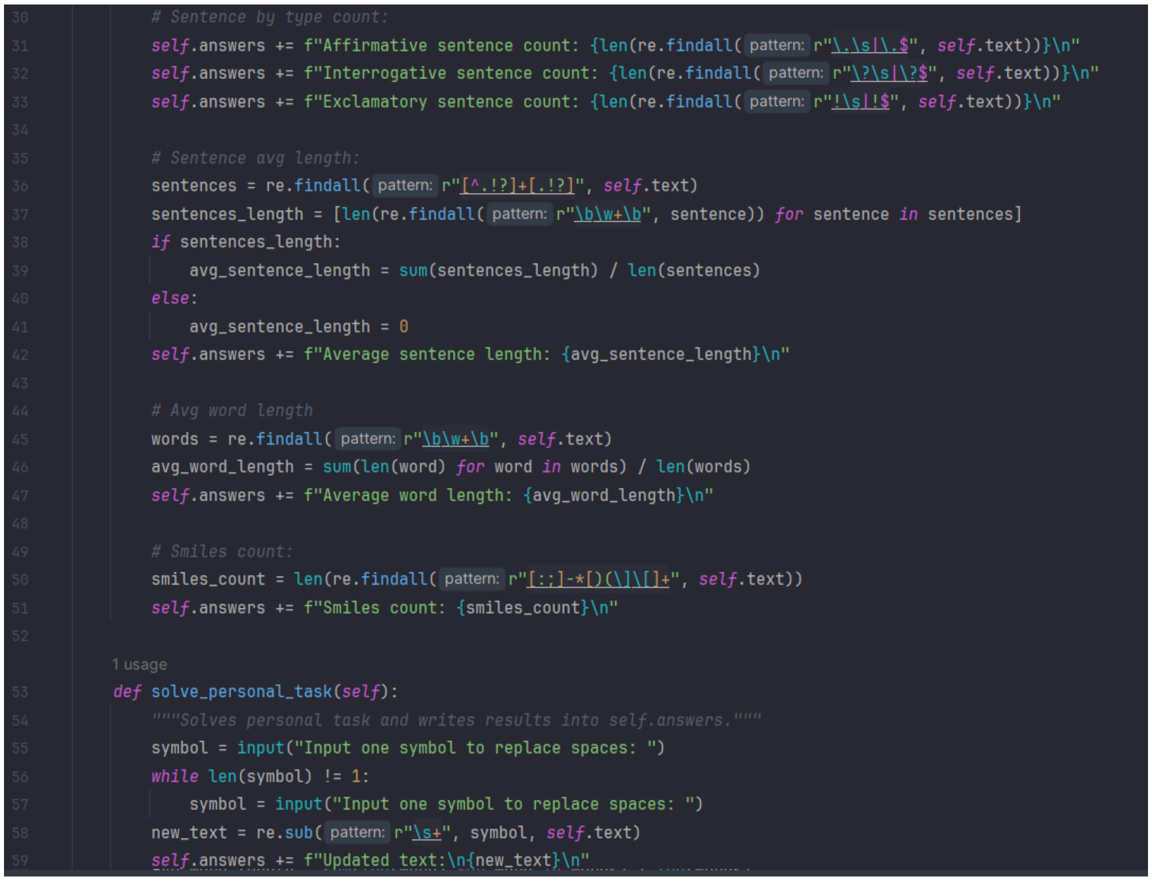
```python
    @staticmethod
    def show_available_commands():
        """Shows list of available commands."""
        print("Available commands:\n/ie - input events\n/c - change task condition\n/s - show events\n/sc - show "
              "events by century\n/cl - show commands list\n/q - to quite task")

    1 usage
    def run(self):
        print(self.desc)
        TaskOne.show_available_commands()
        while True:
            command = input("Command: ")
            if command == "/ie":
                self.input_events()
            elif command == "/c":
                self.input_task_condition()
            elif command == "/s":
                self.show_events()
            elif command == "/sc":
                cent = TaskOne.check_century_validity(input("Enter century: "))
                if cent is None:
                    print("Invalid century.")
                    continue
                self.show_events_by_century(cent)
            elif command == "/cl":
                self.show_available_commands()
            elif command == "/q":
                break
```


```
Command: t1
Task 1: belarusian history events.
Available commands:
/ie - input events
/c - change task condition
/s - show events
/sc - show events by century
/cl - show commands list
/q - to quite task
Command: /s
Date      Event
1863-01-22: Kalinovsky's uprising.
1067-03-03: Battle of the Nemiga River.
1918-03-25: Proclamation of the Belarusian People's Republic.
1410-07-15: Battle of Grunwald.
Command: /sc
Enter century: 20
1918-03-25: Proclamation of the Belarusian People's Republic.
Command: /q
Command list:
```

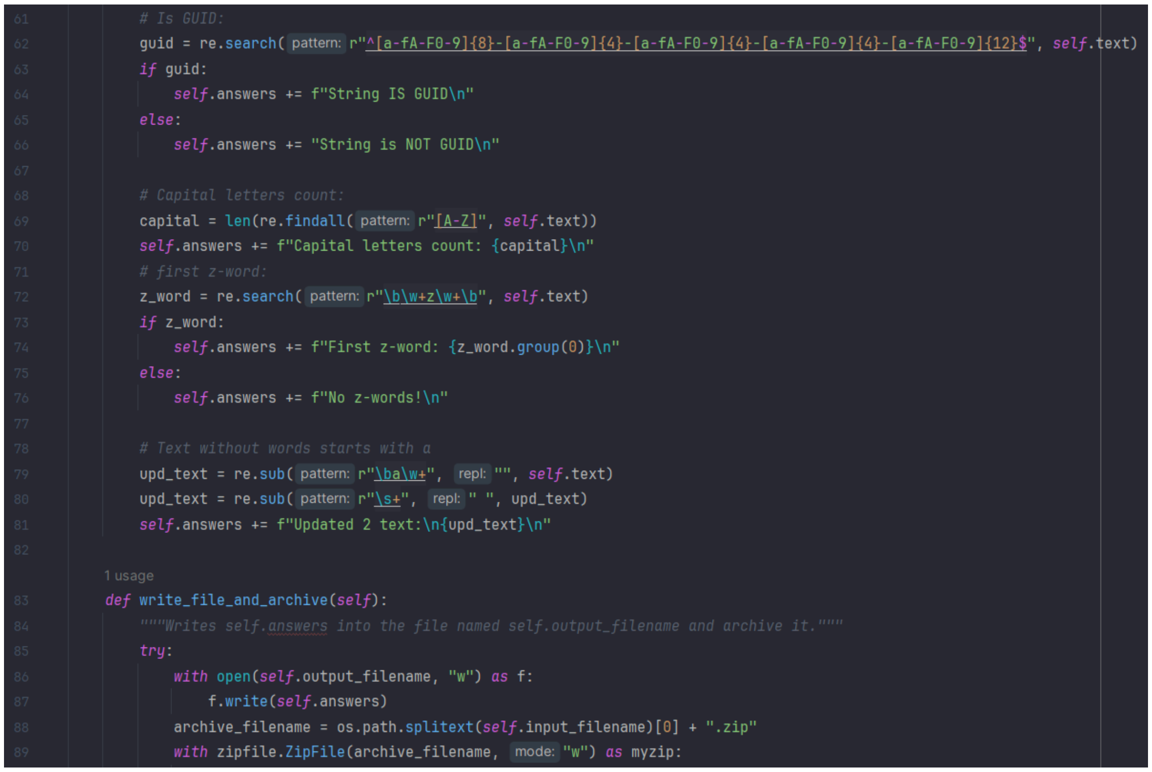
2. Задание 2.



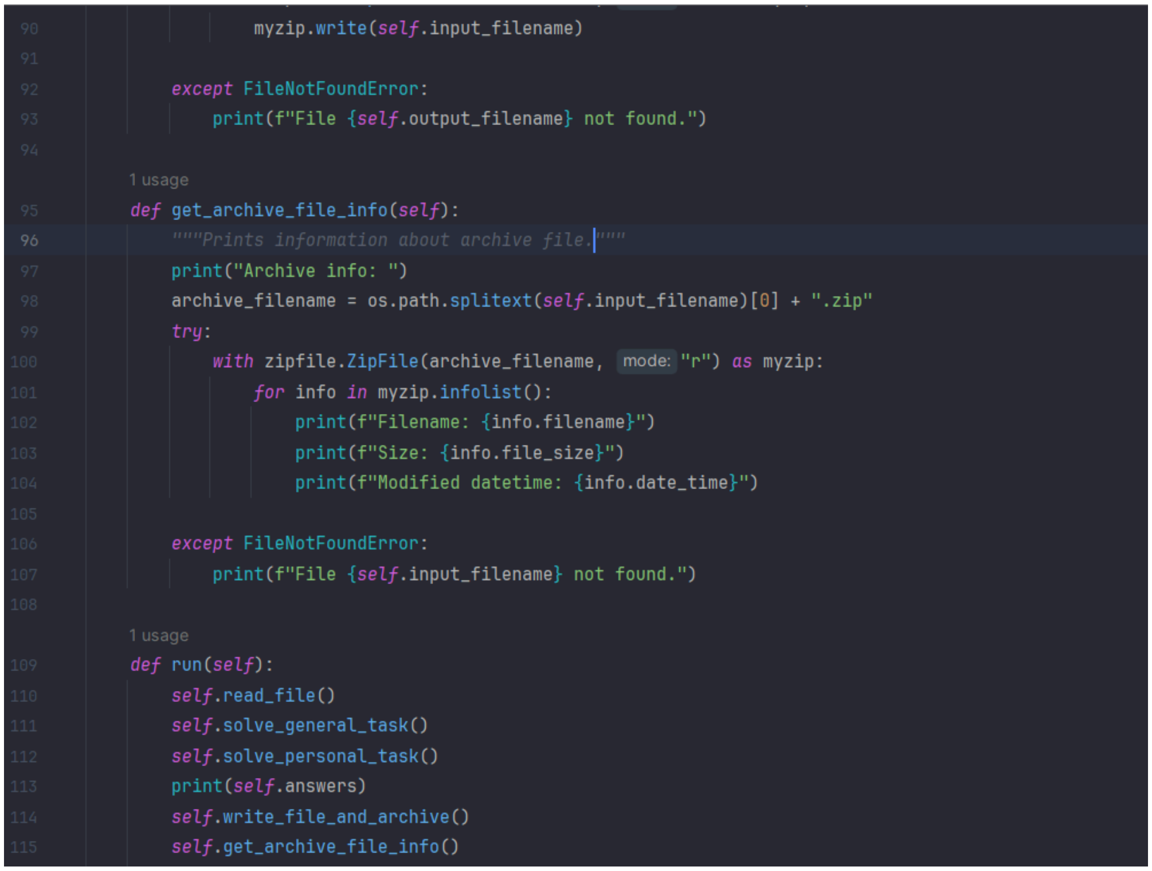
```
import re
import zipfile
import os


1 usage
class TaskTwo:
    def __init__(self, fname="files/task2_text.txt"):
        self.input_filename = fname
        self.output_filename = "files/task2_output.txt"
        self.text = ""
        self.answers = ""

    1 usage
    def read_file(self):
        """Reads entire text from file=input_filename to self.text"""
        try:
            with open(self.input_filename, "r") as f:
                self.text = f.read()
        except FileNotFoundError:
            print(f"File {self.input_filename} not found.")

    1 usage
    def solve_general_task(self):
        """Solves general task and write results into self.answers."""
        # Sentence count:
        match = re.findall( pattern: r"[.!?]\s|[.!?]$", self.text)
        if match:
            self.answers += f"All sentence count: {len(match)}\n"
        else:
            self.answers += "All sentence count: 0\n"
```



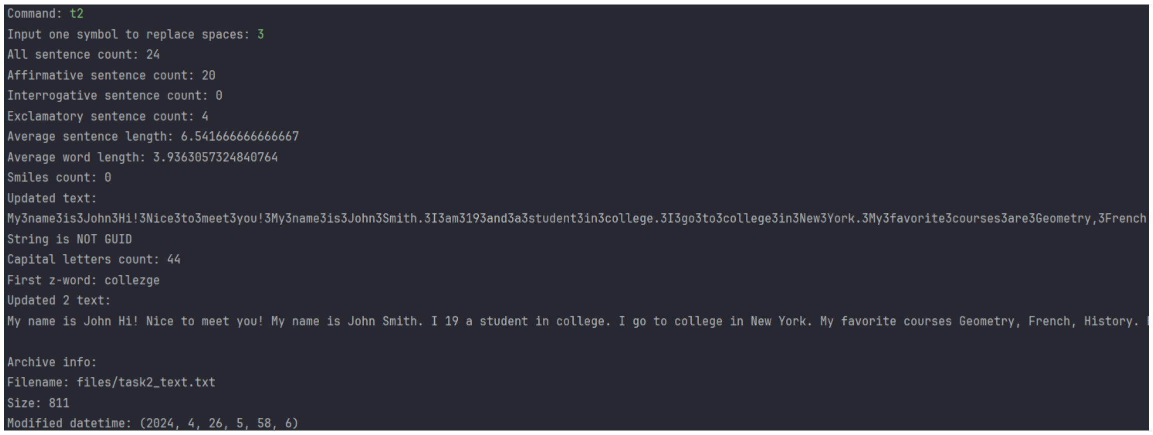
```python
        # Sentence by type count:
        self.answers += f"Affirmative sentence count: {len(re.findall( pattern: r"\.\s|\.$", self.text))}\n"
        self.answers += f"Interrogative sentence count: {len(re.findall( pattern: r"\?\s|\?$", self.text))}\n"
        self.answers += f"Exclamatory sentence count: {len(re.findall( pattern: r"!\s|!$", self.text))}\n"

        # Sentence avg length:
        sentences = re.findall( pattern: r"[^.!?]+[.!?]", self.text)
        sentences_length = [len(re.findall( pattern: r"\b\w+\b", sentence)) for sentence in sentences]
        if sentences_length:
            avg_sentence_length = sum(sentences_length) / len(sentences)
        else:
            avg_sentence_length = 0
        self.answers += f"Average sentence length: {avg_sentence_length}\n"

        # Avg word length
        words = re.findall( pattern: r"\b\w+\b", self.text)
        avg_word_length = sum(len(word) for word in words) / len(words)
        self.answers += f"Average word length: {avg_word_length}\n"

        # Smiles count:
        smiles_count = len(re.findall( pattern: r"[:;]-*[)(\]\[]+", self.text))
        self.answers += f"Smiles count: {smiles_count}\n"

    1 usage
    def solve_personal_task(self):
        """Solves personal task and writes results into self.answers."""
        symbol = input("Input one symbol to replace spaces: ")
        while len(symbol) != 1:
            symbol = input("Input one symbol to replace spaces: ")
        new_text = re.sub( pattern: r"\s+", symbol, self.text)
        self.answers += f"Updated text:\n{new_text}\n"
```



```python
        # Is GUID:
        guid = re.search( pattern: r"^[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}$", self.text)
        if guid:
            self.answers += f"String IS GUID\n"
        else:
            self.answers += "String is NOT GUID\n"

        # Capital letters count:
        capital = len(re.findall( pattern: r"[A-Z]", self.text))
        self.answers += f"Capital letters count: {capital}\n"
        # first z-word:
        z_word = re.search( pattern: r"\b\w+z\w+\b", self.text)
        if z_word:
            self.answers += f"First z-word: {z_word.group(0)}\n"
        else:
            self.answers += f"No z-words!\n"

        # Text without words starts with a
        upd_text = re.sub( pattern: r"\ba\w+", repl: "", self.text)
        upd_text = re.sub( pattern: r"\s+", repl: " ", upd_text)
        self.answers += f"Updated 2 text:\n{upd_text}\n"

    1 usage
    def write_file_and_archive(self):
        """Writes self.answers into the file named self.output_filename and archive it."""
        try:
            with open(self.output_filename, "w") as f:
                f.write(self.answers)
            archive_filename = os.path.splitext(self.input_filename)[0] + ".zip"
            with zipfile.ZipFile(archive_filename,  mode: "w") as myzip:
```

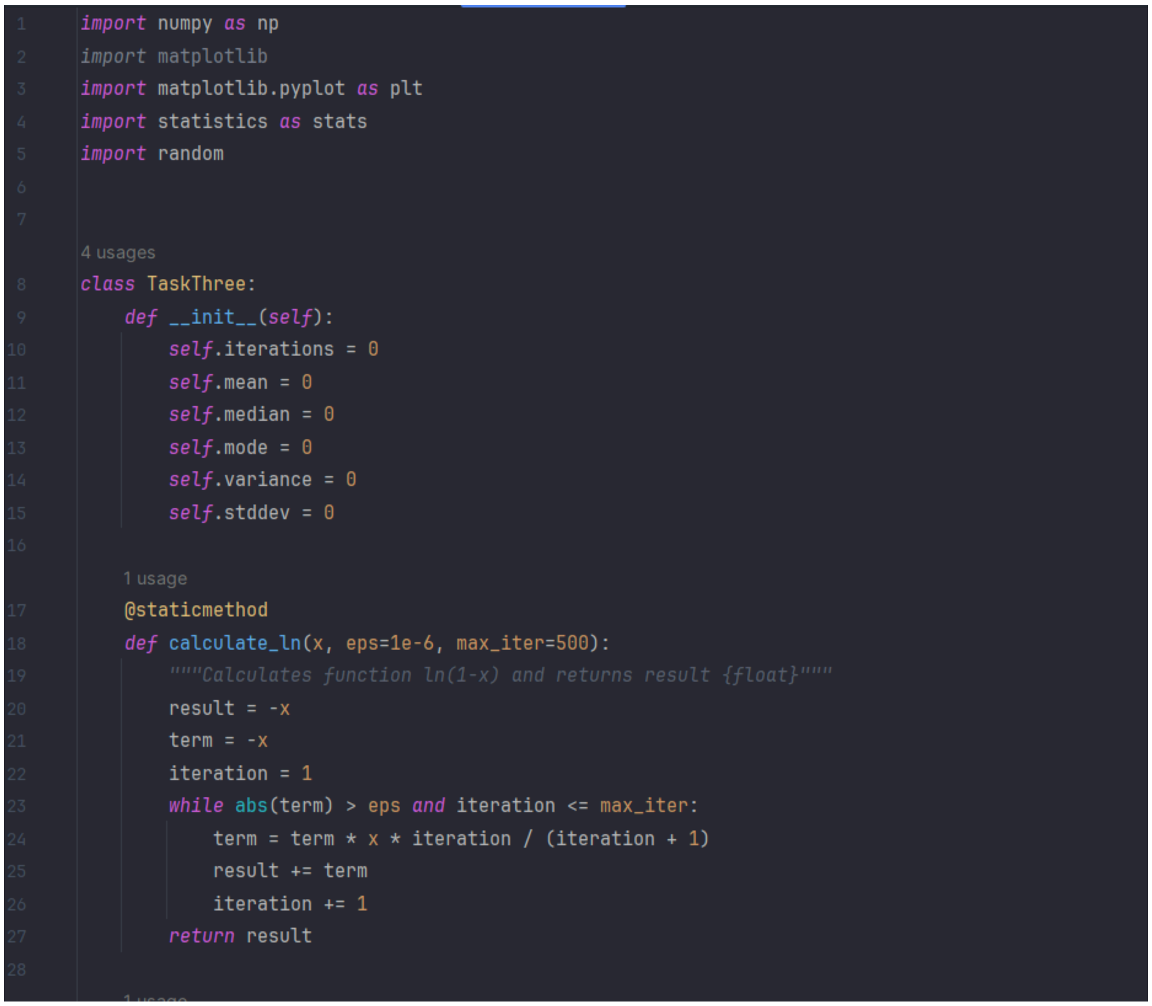


```python
                myzip.write(self.input_filename)

        except FileNotFoundError:
            print(f"File {self.output_filename} not found.")

    def get_archive_file_info(self):
        """Prints information about archive file."""
        print("Archive info: ")
        archive_filename = os.path.splitext(self.input_filename)[0] + ".zip"
        try:
            with zipfile.ZipFile(archive_filename,  mode: "r") as myzip:
                for info in myzip.infolist():
                    print(f"Filename: {info.filename}")
                    print(f"Size: {info.file_size}")
                    print(f"Modified datetime: {info.date_time}")

        except FileNotFoundError:
            print(f"File {self.input_filename} not found.")

    def run(self):
        self.read_file()
        self.solve_general_task()
        self.solve_personal_task()
        print(self.answers)
        self.write_file_and_archive()
        self.get_archive_file_info()
```

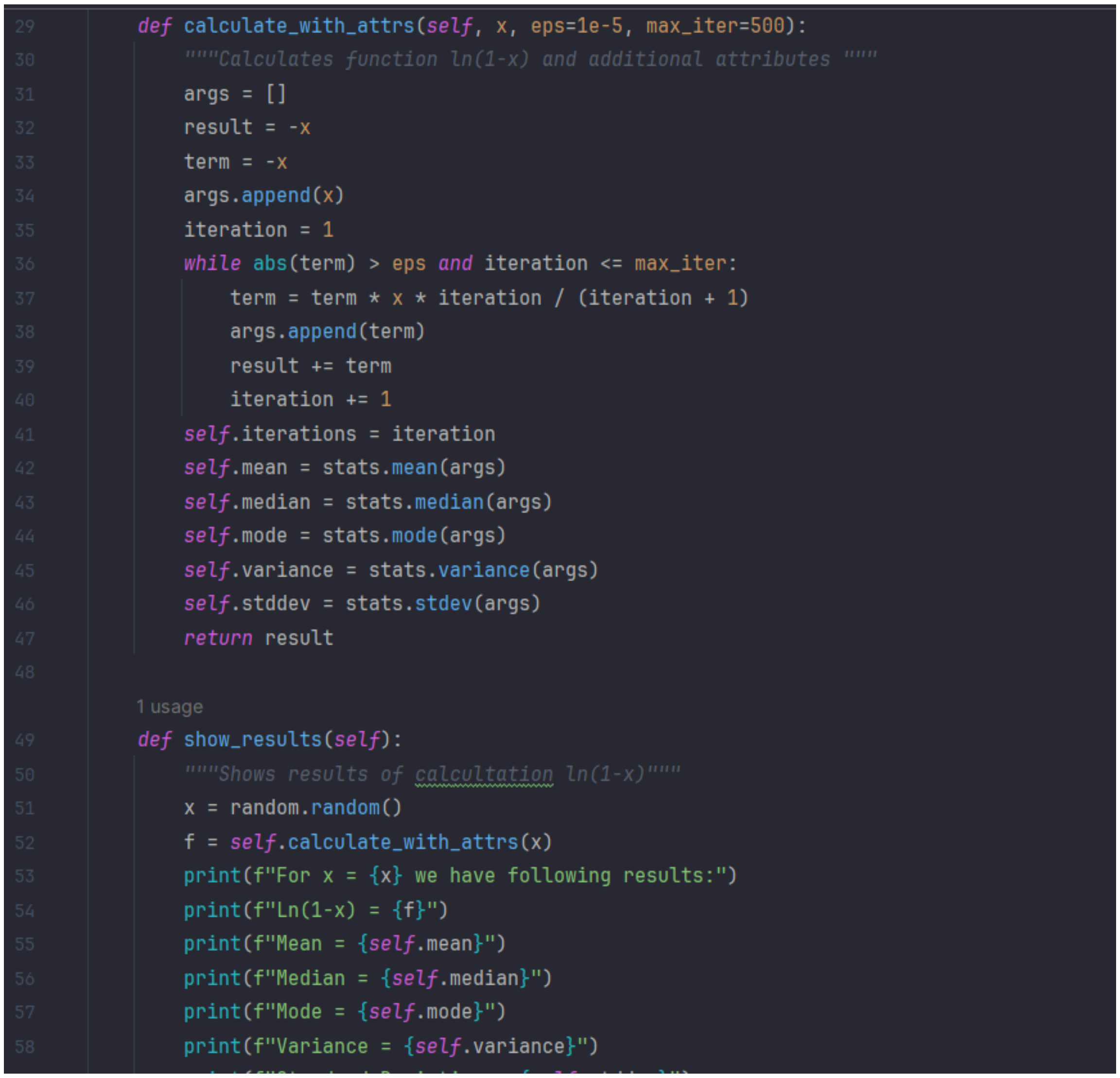


```
Command: t2
Input one symbol to replace spaces: 3
All sentence count: 24
Affirmative sentence count: 20
Interrogative sentence count: 0
Exclamatory sentence count: 4
Average sentence length: 6.541666666666667
Average word length: 3.9363057324840764
Smiles count: 0
Updated text:
My3name3is3John3Hi!3Nice3to3meet3you!3My3name3is3John3Smith.3I3am3193and3a3student3in3college.3I3go3to3college3in3New3York.3My3favorite3courses3are3Geometry,3French
String is NOT GUID
Capital letters count: 44
First z-word: collezge
Updated 2 text:
My name is John Hi! Nice to meet you! My name is John Smith. I 19 a student in college. I go to college in New York. My favorite courses Geometry, French, History.

Archive info:
Filename: files/task2_text.txt
Size: 811
Modified datetime: (2024, 4, 26, 5, 58, 6)
```

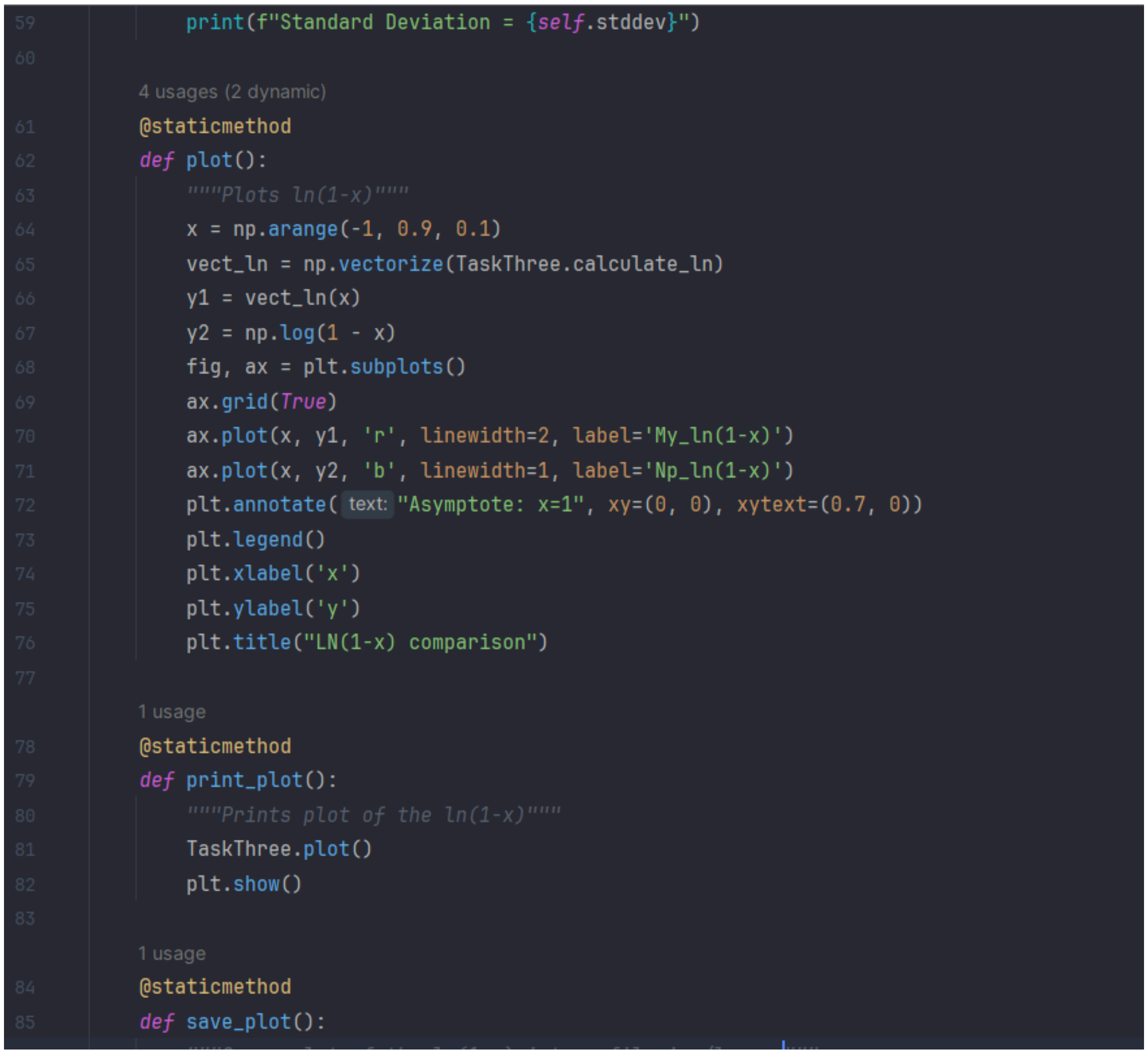
3. Задание 3



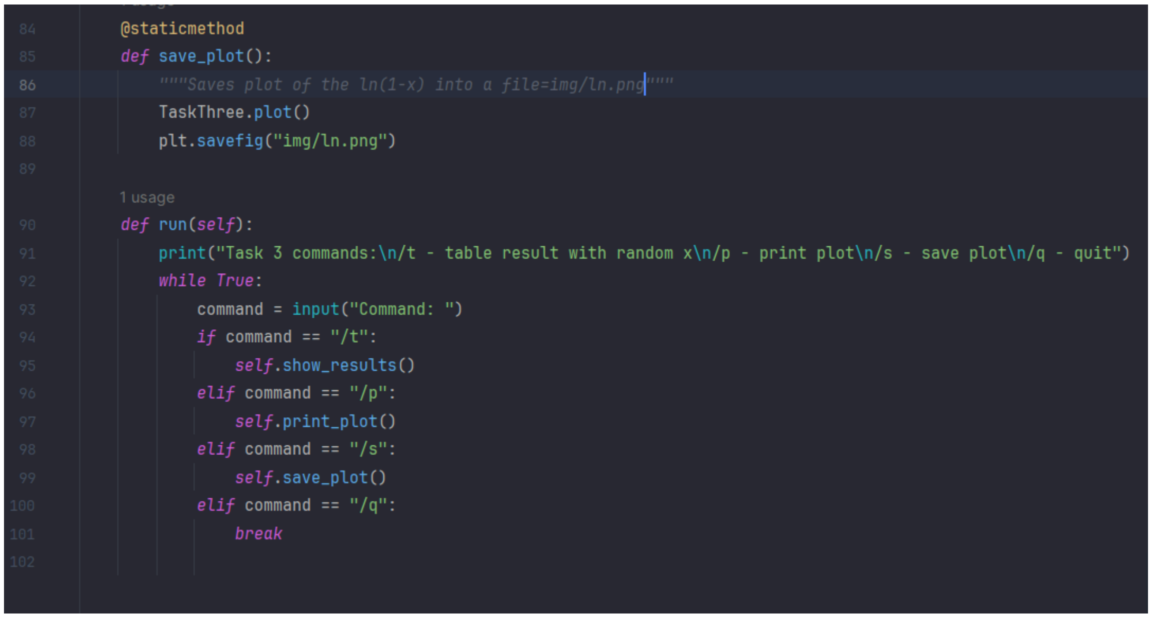
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import statistics as stats
import random


4 usages
class TaskThree:
    def __init__(self):
        self.iterations = 0
        self.mean = 0
        self.median = 0
        self.mode = 0
        self.variance = 0
        self.stddev = 0

    1 usage
    @staticmethod
    def calculate_ln(x, eps=1e-6, max_iter=500):
        """Calculates function ln(1-x) and returns result {float}"""
        result = -x
        term = -x
        iteration = 1
        while abs(term) > eps and iteration <= max_iter:
            term = term * x * iteration / (iteration + 1)
            result += term
            iteration += 1
        return result

```



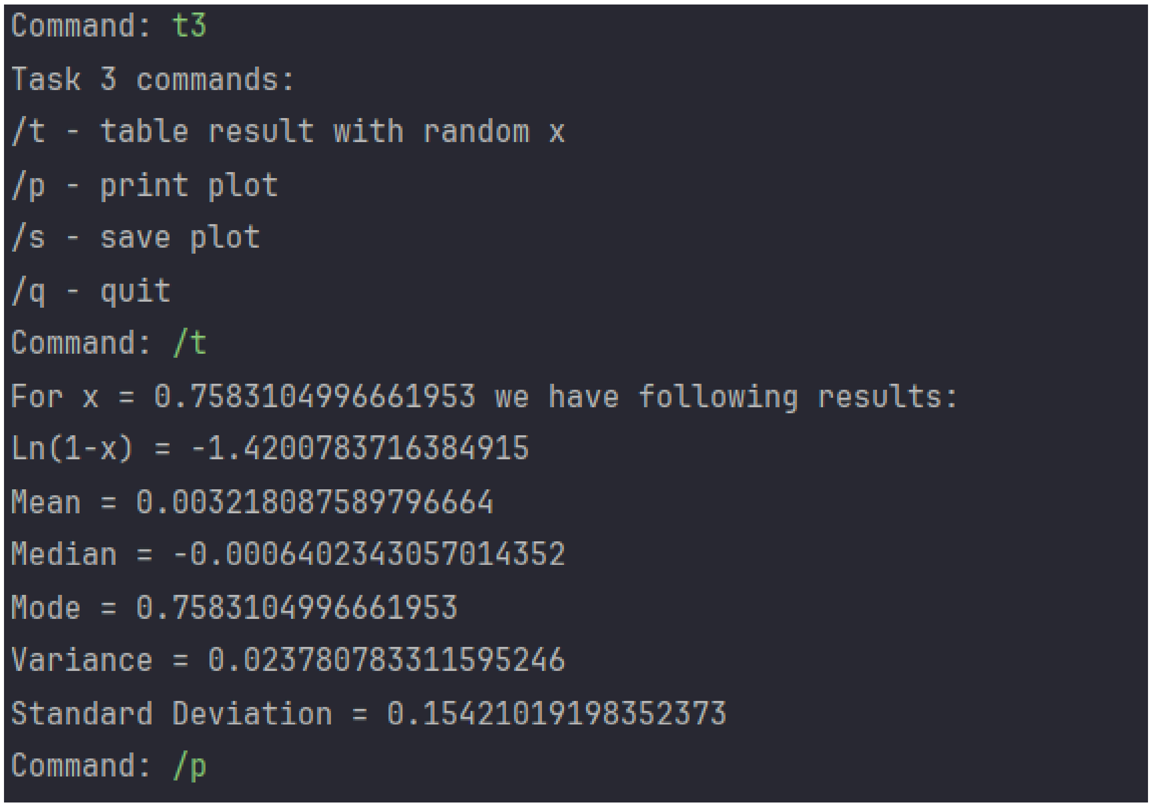
```python
    def calculate_with_attrs(self, x, eps=1e-5, max_iter=500):
        """Calculates function ln(1-x) and additional attributes """
        args = []
        result = -x
        term = -x
        args.append(x)
        iteration = 1
        while abs(term) > eps and iteration <= max_iter:
            term = term * x * iteration / (iteration + 1)
            args.append(term)
            result += term
            iteration += 1
        self.iterations = iteration
        self.mean = stats.mean(args)
        self.median = stats.median(args)
        self.mode = stats.mode(args)
        self.variance = stats.variance(args)
        self.stddev = stats.stdev(args)
        return result

    1 usage
    def show_results(self):
        """Shows results of calcultation ln(1-x)"""
        x = random.random()
        f = self.calculate_with_attrs(x)
        print(f"For x = {x} we have following results:")
        print(f"Ln(1-x) = {f}")
        print(f"Mean = {self.mean}")
        print(f"Median = {self.median}")
        print(f"Mode = {self.mode}")
        print(f"Variance = {self.variance}")
```



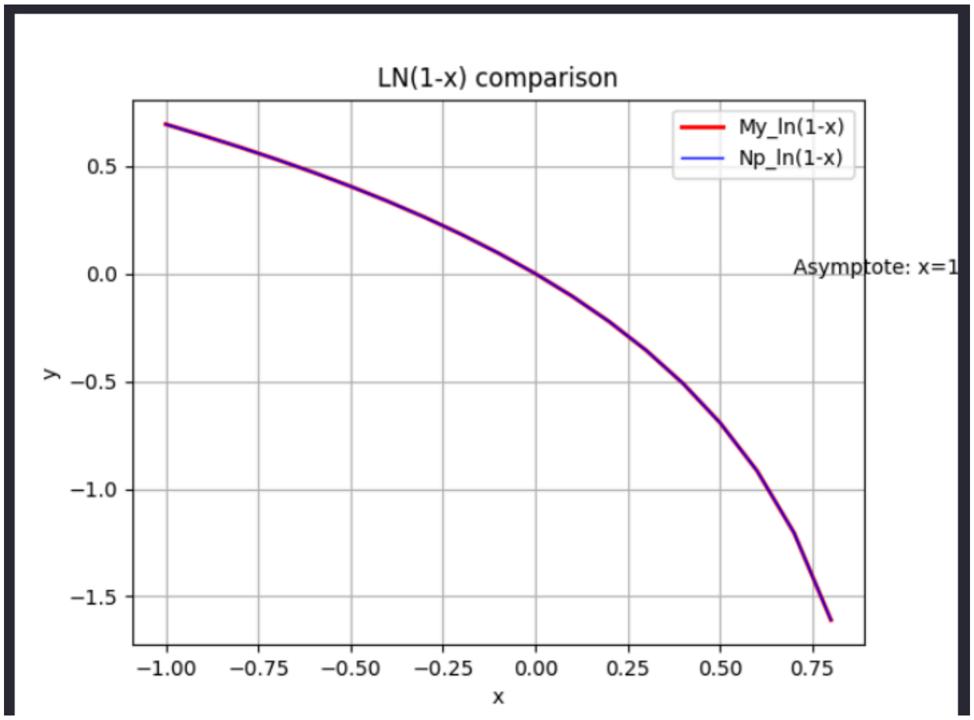
```python
        print(f"Standard Deviation = {self.stddev}")

    4 usages (2 dynamic)
    @staticmethod
    def plot():
        """Plots ln(1-x)"""
        x = np.arange(-1, 0.9, 0.1)
        vect_ln = np.vectorize(TaskThree.calculate_ln)
        y1 = vect_ln(x)
        y2 = np.log(1 - x)
        fig, ax = plt.subplots()
        ax.grid(True)
        ax.plot(x, y1, 'r', linewidth=2, label='My_ln(1-x)')
        ax.plot(x, y2, 'b', linewidth=1, label='Np_ln(1-x)')
        plt.annotate( text: "Asymptote: x=1", xy=(0, 0), xytext=(0.7, 0))
        plt.legend()
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title("LN(1-x) comparison")

    1 usage
    @staticmethod
    def print_plot():
        """Prints plot of the ln(1-x)"""
        TaskThree.plot()
        plt.show()

    1 usage
    @staticmethod
    def save_plot():
```



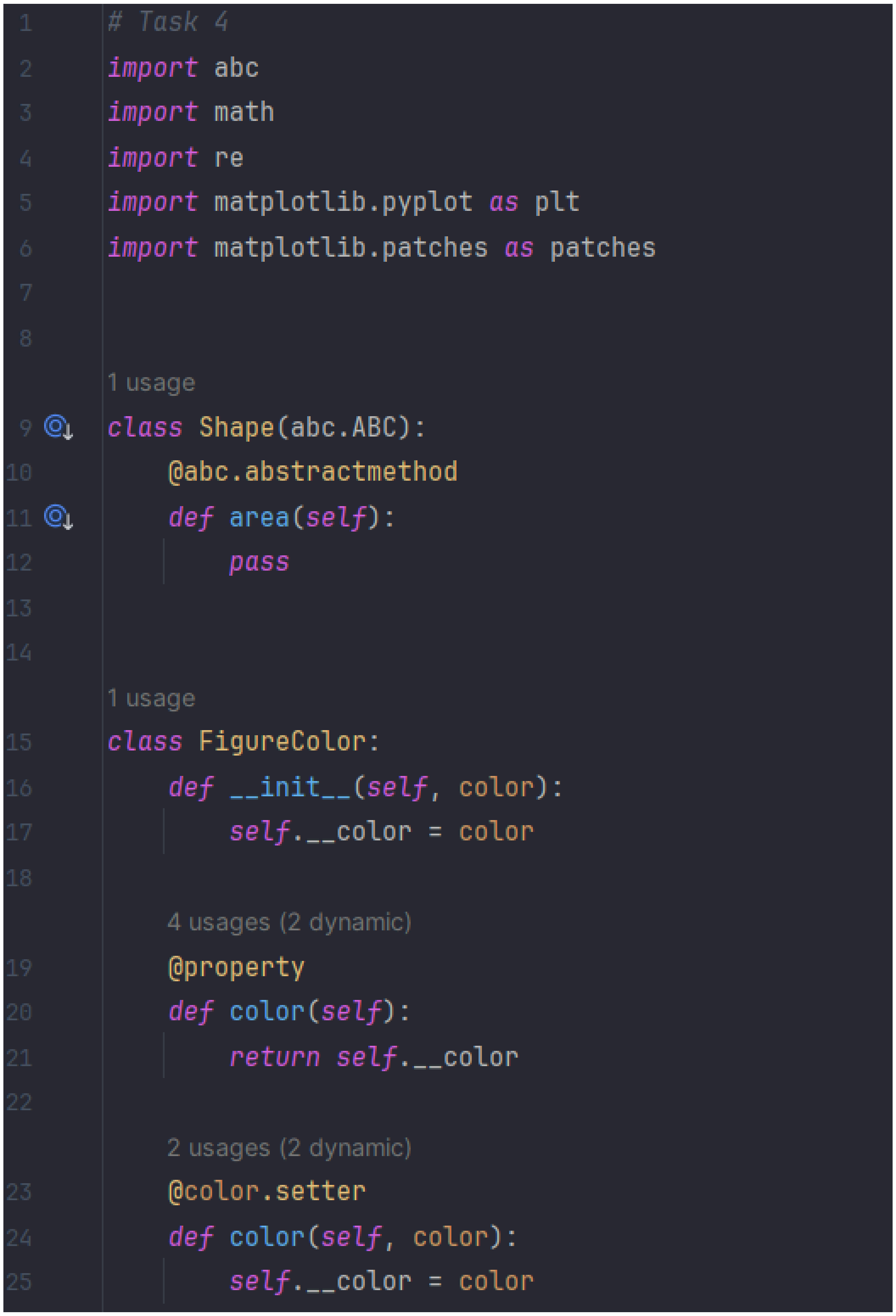
```python
    @staticmethod
    def save_plot():
        """Saves plot of the ln(1-x) into a file=img/ln.png"""
        TaskThree.plot()
        plt.savefig("img/ln.png")

    1 usage
    def run(self):
        print("Task 3 commands:\n/t - table result with random x\n/p - print plot\n/s - save plot\n/q - quit")
        while True:
            command = input("Command: ")
            if command == "/t":
                self.show_results()
            elif command == "/p":
                self.print_plot()
            elif command == "/s":
                self.save_plot()
            elif command == "/q":
                break
```



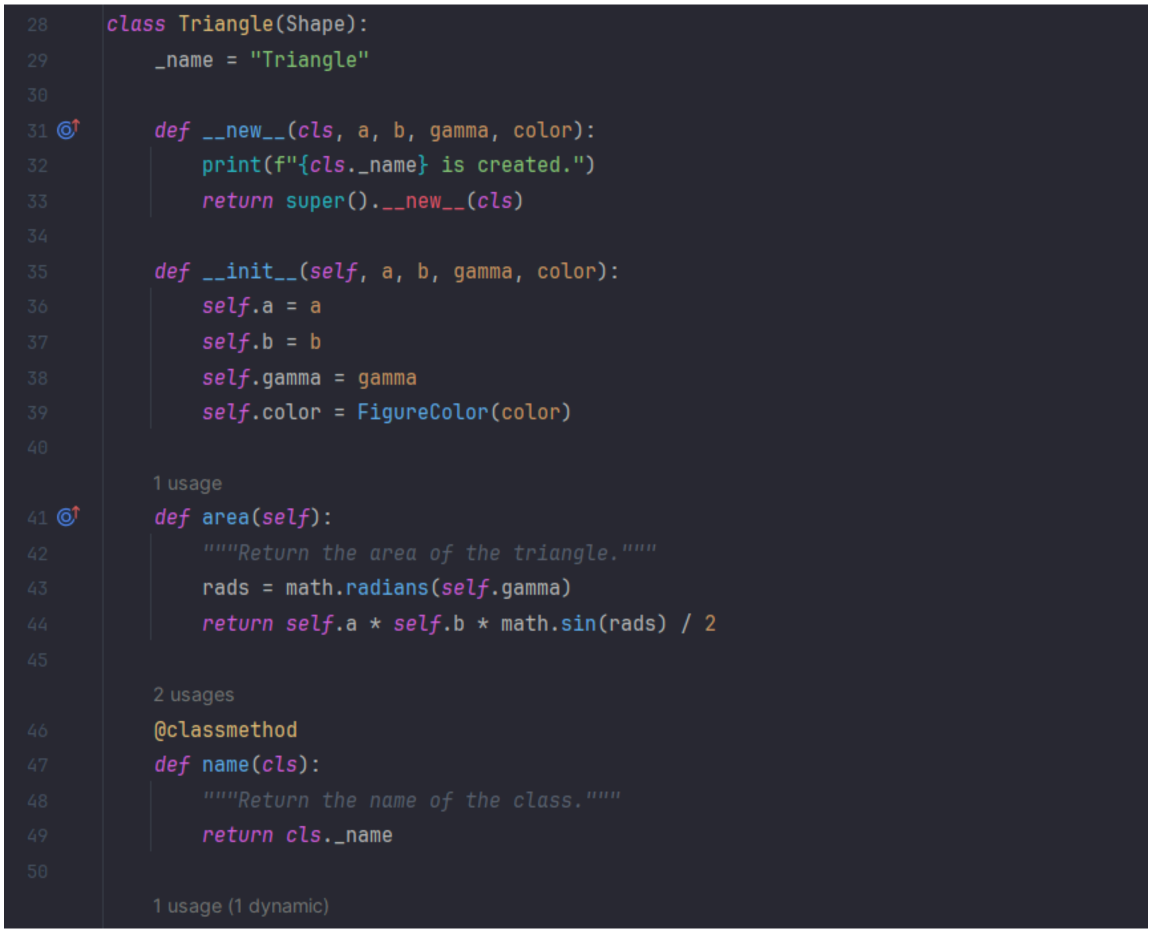
```
Command: t3
Task 3 commands:
/t - table result with random x
/p - print plot
/s - save plot
/q - quit
Command: /t
For x = 0.7583104996661953 we have following results:
Ln(1-x) = -1.4200783716384915
Mean = 0.003218087589796664
Median = -0.0006402343057014352
Mode = 0.7583104996661953
Variance = 0.023780783311595246
Standard Deviation = 0.15421019198352373
Command: /p
```



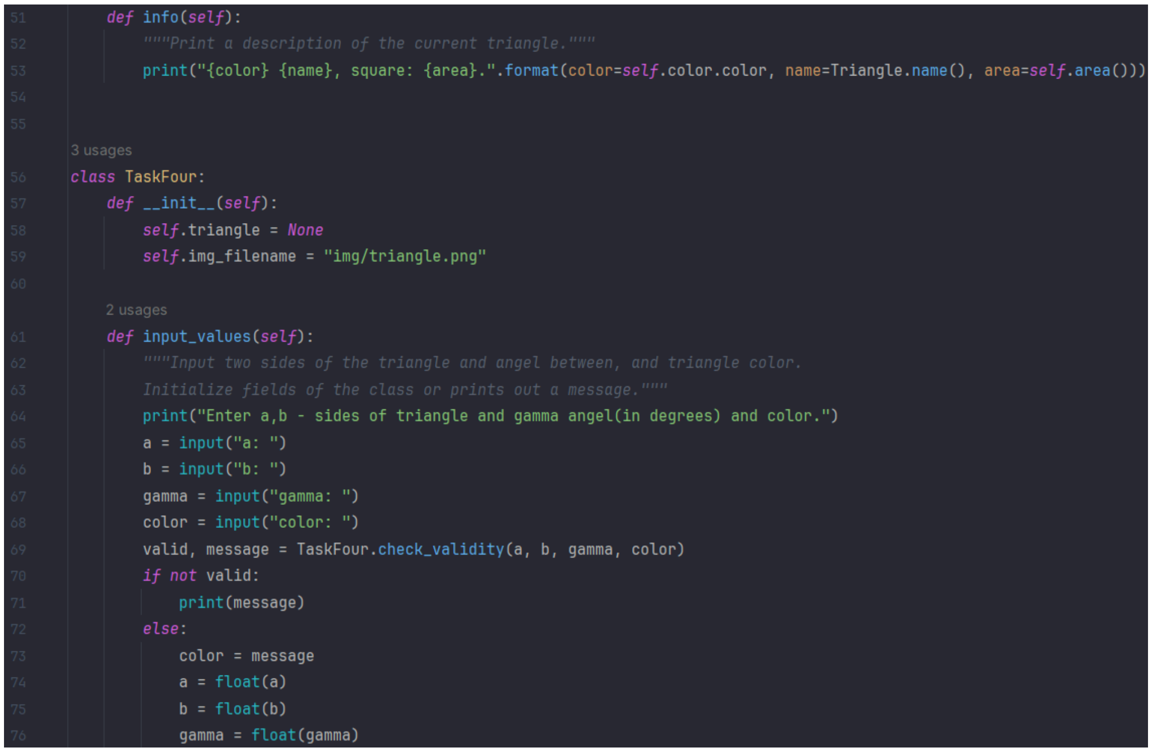
4. Задание 4



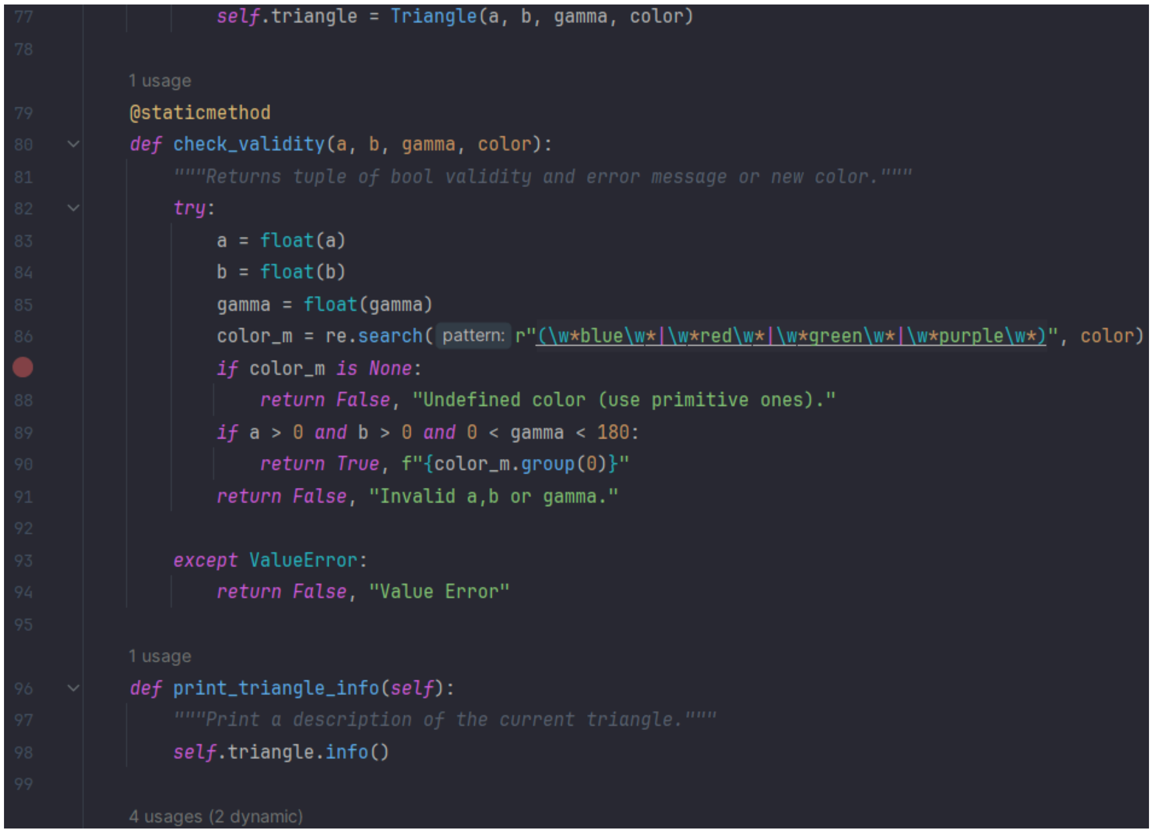
```python
# Task 4
import abc
import math
import re
import matplotlib.pyplot as plt
import matplotlib.patches as patches


1 usage
class Shape(abc.ABC):
    @abc.abstractmethod
    def area(self):
        pass


1 usage
class FigureColor:
    def __init__(self, color):
        self.__color = color

    4 usages (2 dynamic)
    @property
    def color(self):
        return self.__color

    2 usages (2 dynamic)
    @color.setter
    def color(self, color):
        self.__color = color
```



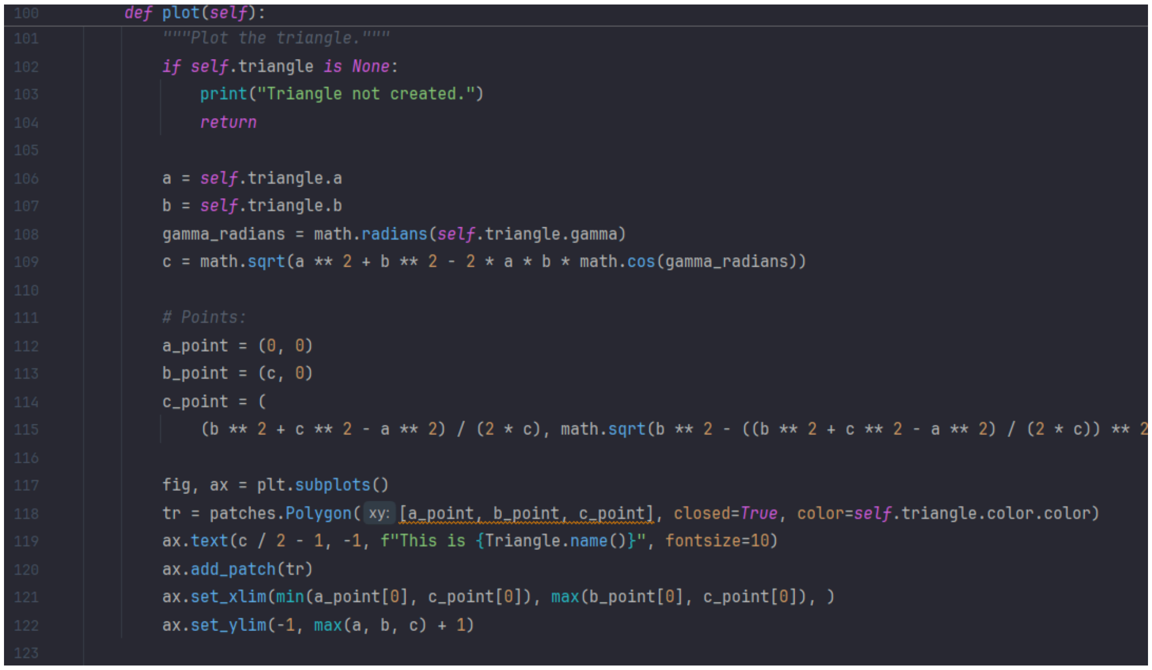
```python
class Triangle(Shape):
    _name = "Triangle"

    def __new__(cls, a, b, gamma, color):
        print(f"{cls._name} is created.")
        return super().__new__(cls)

    def __init__(self, a, b, gamma, color):
        self.a = a
        self.b = b
        self.gamma = gamma
        self.color = FigureColor(color)

    1 usage
    def area(self):
        """Return the area of the triangle."""
        rads = math.radians(self.gamma)
        return self.a * self.b * math.sin(rads) / 2

    2 usages
    @classmethod
    def name(cls):
        """Return the name of the class."""
        return cls._name

    1 usage (1 dynamic)
```



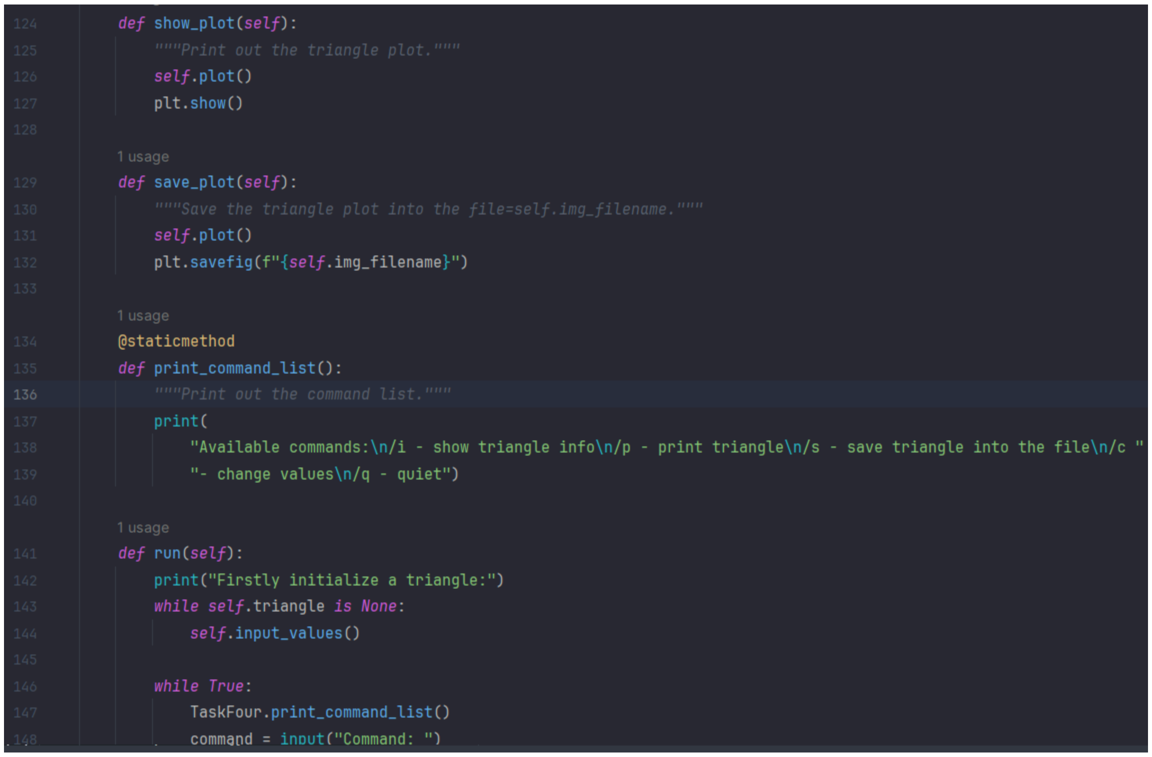
```python
    def info(self):
        """Print a description of the current triangle."""
        print("{color} {name}, square: {area}.".format(color=self.color.color, name=Triangle.name(), area=self.area()))


3 usages
class TaskFour:
    def __init__(self):
        self.triangle = None
        self.img_filename = "img/triangle.png"

    2 usages
    def input_values(self):
        """Input two sides of the triangle and angel between, and triangle color.
        Initialize fields of the class or prints out a message."""
        print("Enter a,b - sides of triangle and gamma angel(in degrees) and color.")
        a = input("a: ")
        b = input("b: ")
        gamma = input("gamma: ")
        color = input("color: ")
        valid, message = TaskFour.check_validity(a, b, gamma, color)
        if not valid:
            print(message)
        else:
            color = message
            a = float(a)
            b = float(b)
            gamma = float(gamma)
```



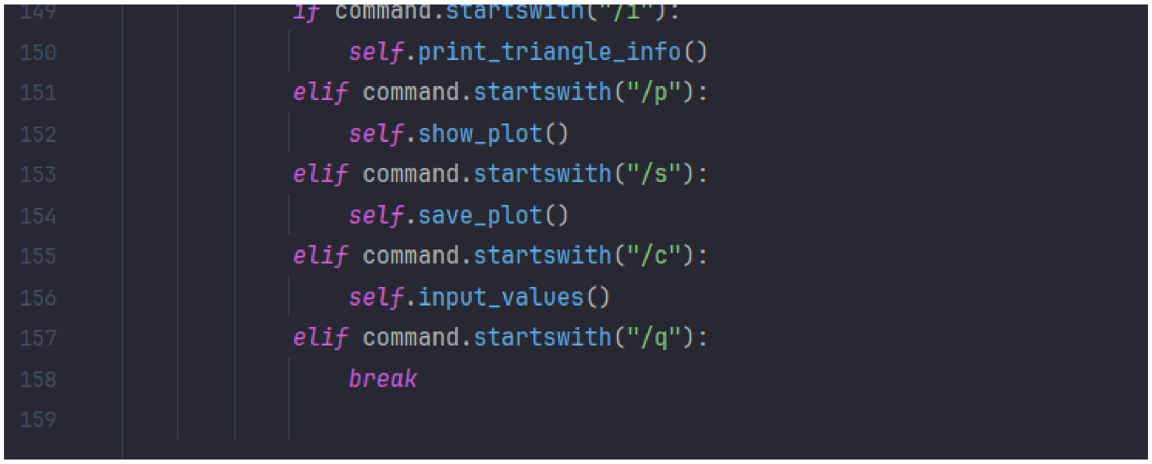
```
            self.triangle = Triangle(a, b, gamma, color)

    1 usage
    @staticmethod
    def check_validity(a, b, gamma, color):
        """Returns tuple of bool validity and error message or new color."""
        try:
            a = float(a)
            b = float(b)
            gamma = float(gamma)
            color_m = re.search( pattern: r"(\w*blue\w*|\w*red\w*|\w*green\w*|\w*purple\w*)", color)
            if color_m is None:
                return False, "Undefined color (use primitive ones)."
            if a > 0 and b > 0 and 0 < gamma < 180:
                return True, f"{color_m.group(0)}"
            return False, "Invalid a,b or gamma."

        except ValueError:
            return False, "Value Error"

    1 usage
    def print_triangle_info(self):
        """Print a description of the current triangle."""
        self.triangle.info()

    4 usages (2 dynamic)
```



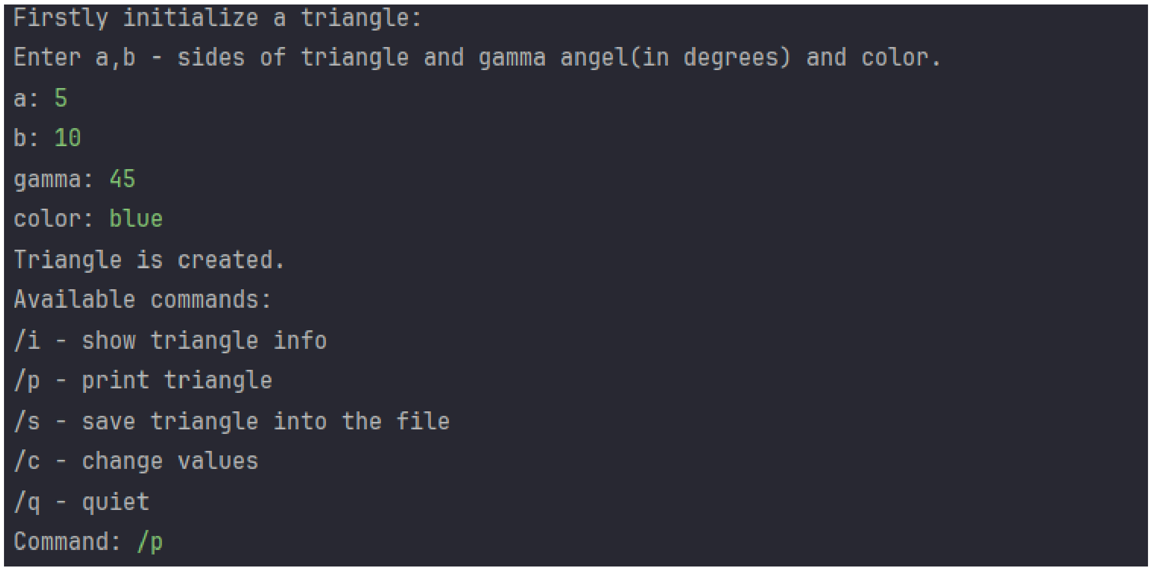
```
    def plot(self):
        """Plot the triangle."""
        if self.triangle is None:
            print("Triangle not created.")
            return

        a = self.triangle.a
        b = self.triangle.b
        gamma_radians = math.radians(self.triangle.gamma)
        c = math.sqrt(a ** 2 + b ** 2 - 2 * a * b * math.cos(gamma_radians))

        # Points:
        a_point = (0, 0)
        b_point = (c, 0)
        c_point = (
            (b ** 2 + c ** 2 - a ** 2) / (2 * c), math.sqrt(b ** 2 - ((b ** 2 + c ** 2 - a ** 2) / (2 * c)) ** 2

        fig, ax = plt.subplots()
        tr = patches.Polygon( xy: [a_point, b_point, c_point], closed=True, color=self.triangle.color.color)
        ax.text(c / 2 - 1, -1, f"This is {Triangle.name()}", fontsize=10)
        ax.add_patch(tr)
        ax.set_xlim(min(a_point[0], c_point[0]), max(b_point[0], c_point[0]), )
        ax.set_ylim(-1, max(a, b, c) + 1)

```



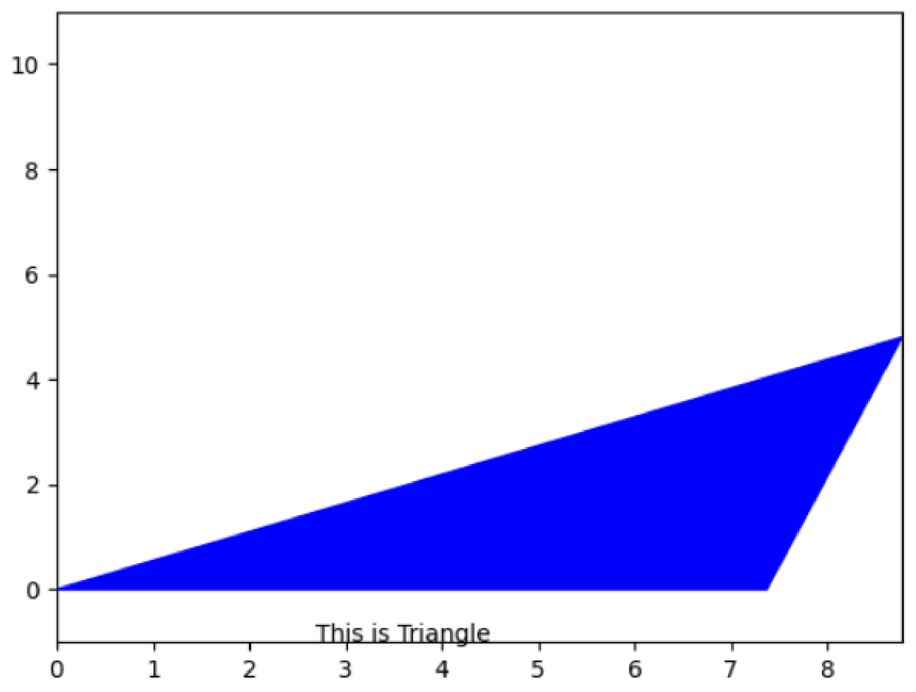
```python
    def show_plot(self):
        """Print out the triangle plot."""
        self.plot()
        plt.show()

    1 usage
    def save_plot(self):
        """Save the triangle plot into the file=self.img_filename."""
        self.plot()
        plt.savefig(f"{self.img_filename}")

    1 usage
    @staticmethod
    def print_command_list():
        """Print out the command list."""
        print(
            "Available commands:\n/i - show triangle info\n/p - print triangle\n/s - save triangle into the file\n/c "
            "- change values\n/q - quiet")

    1 usage
    def run(self):
        print("Firstly initialize a triangle:")
        while self.triangle is None:
            self.input_values()

        while True:
            TaskFour.print_command_list()
            command = input("Command: ")
```



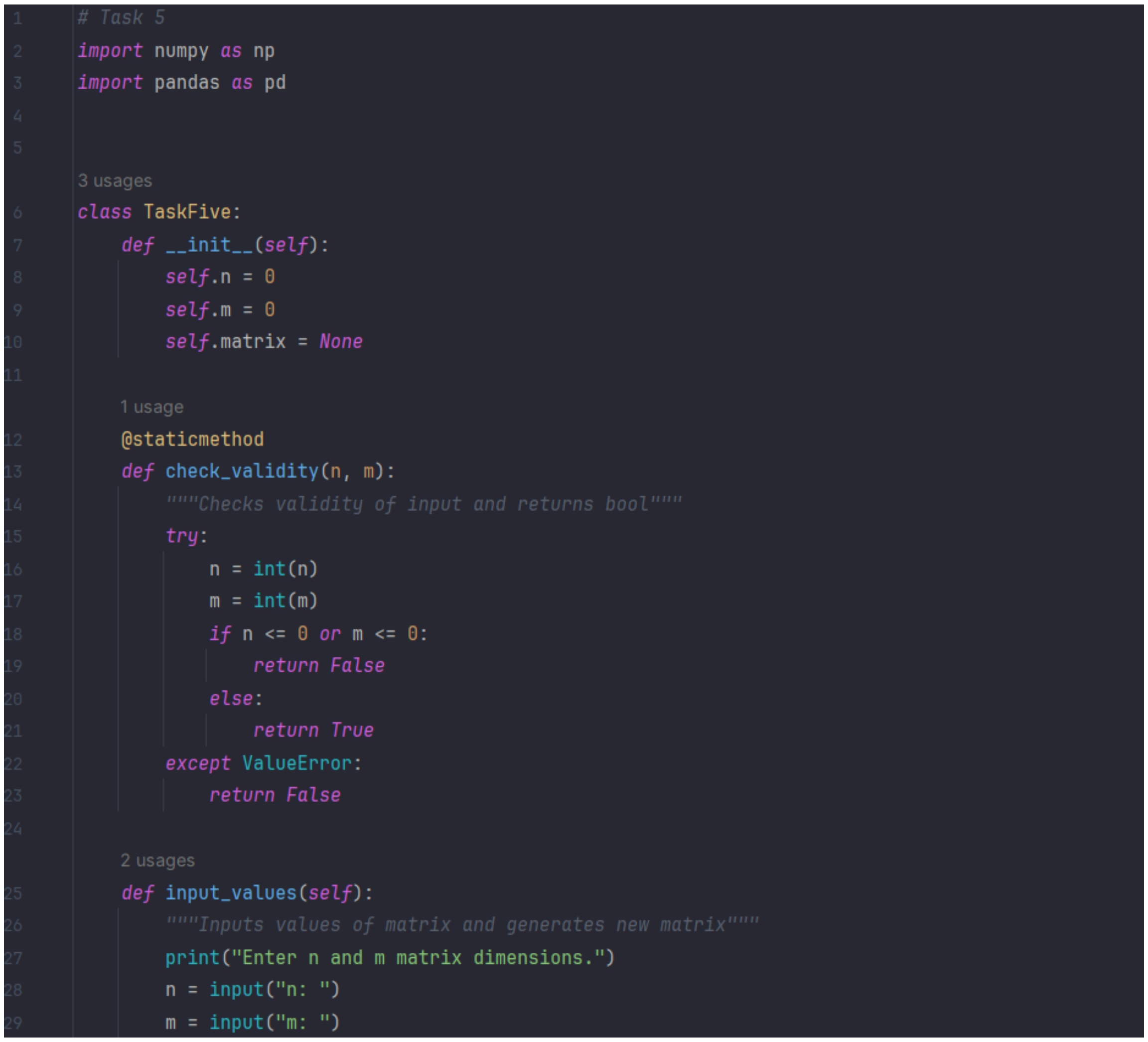
```python
            if command.startswith("/i"):
                self.print_triangle_info()
            elif command.startswith("/p"):
                self.show_plot()
            elif command.startswith("/s"):
                self.save_plot()
            elif command.startswith("/c"):
                self.input_values()
            elif command.startswith("/q"):
                break

```



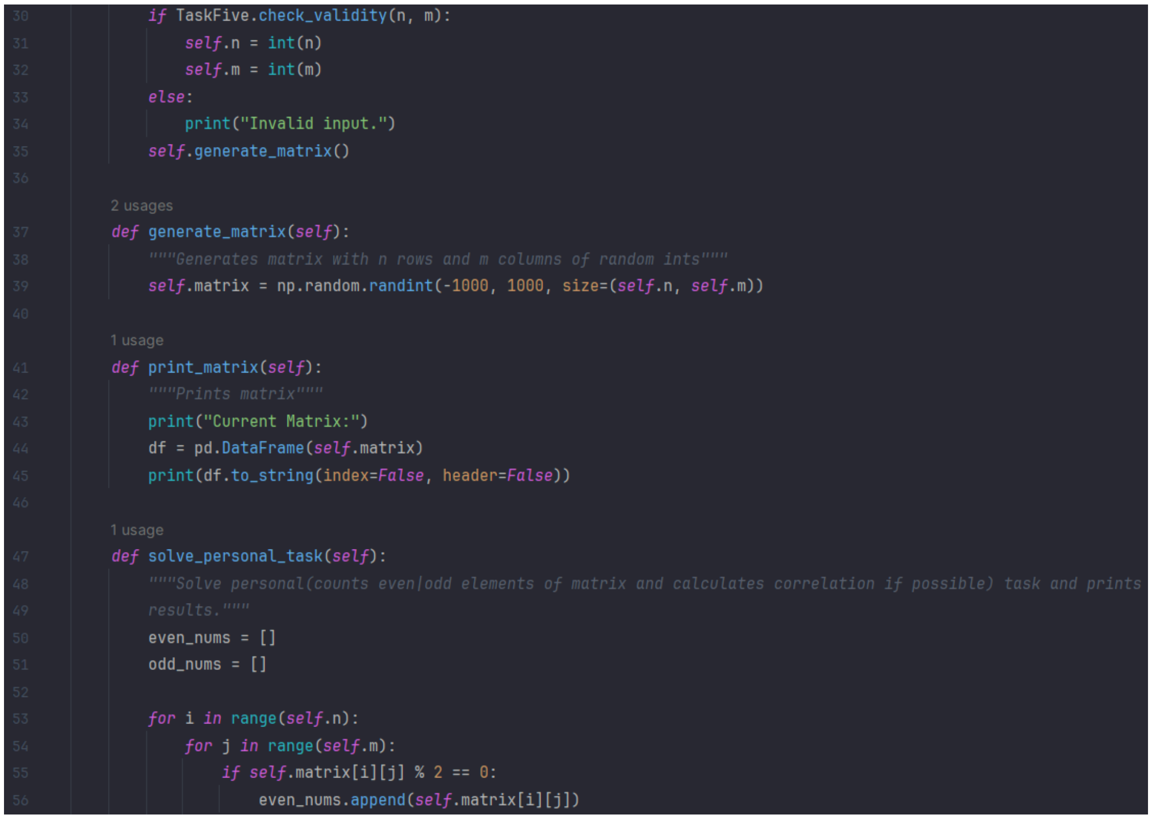
```
Firstly initialize a triangle:
Enter a,b - sides of triangle and gamma angel(in degrees) and color.
a: 5
b: 10
gamma: 45
color: blue
Triangle is created.
Available commands:
/i - show triangle info
/p - print triangle
/s - save triangle into the file
/c - change values
/q - quiet
Command: /p
```



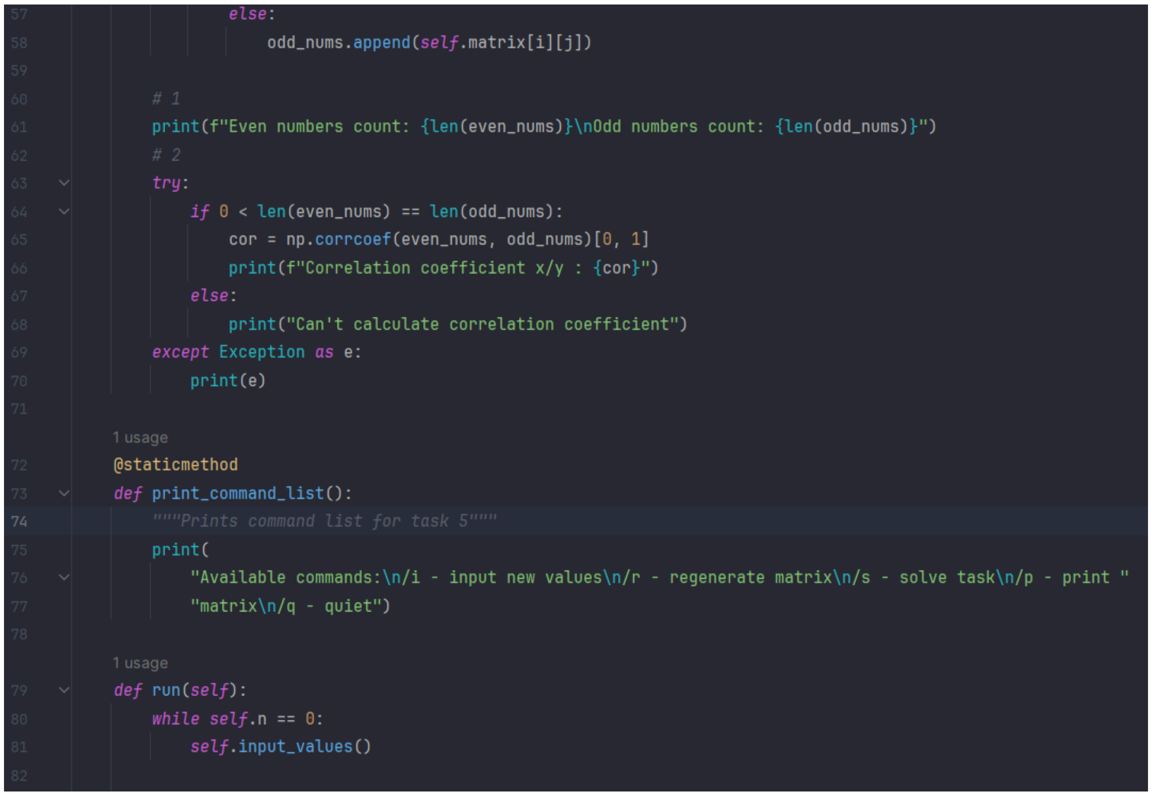
5. Задание 5



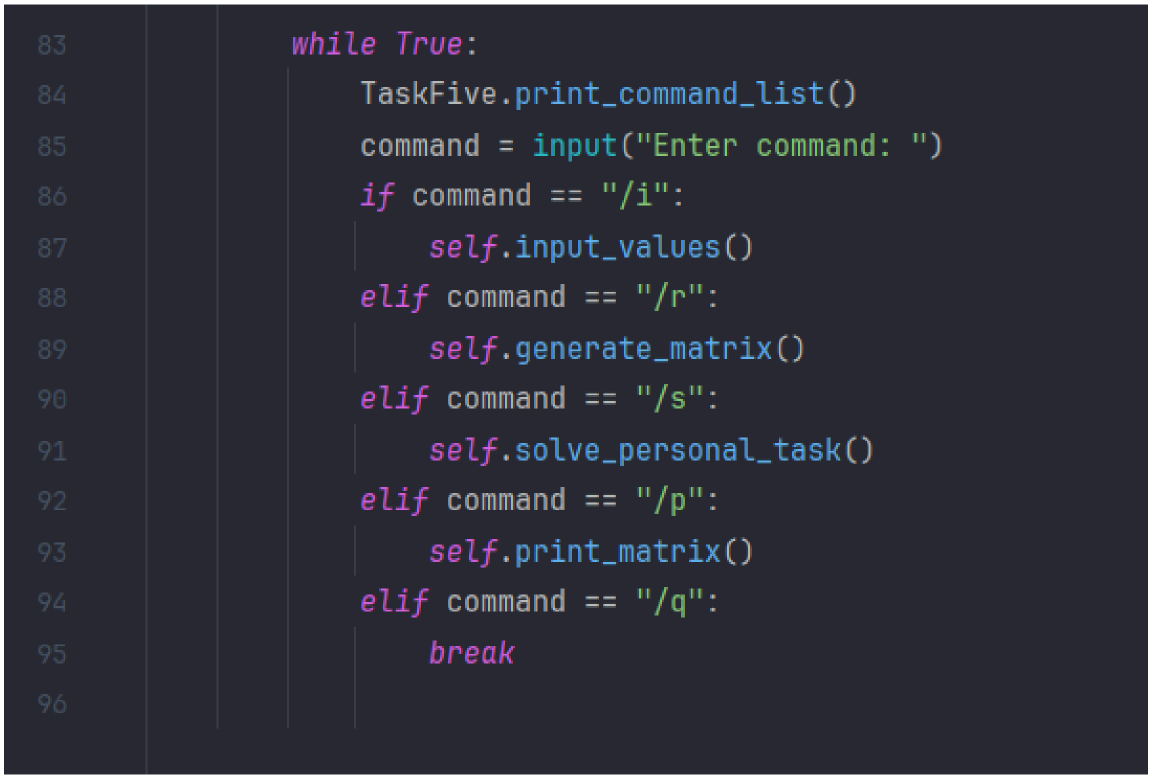
```
# Task 5
import numpy as np
import pandas as pd


3 usages
class TaskFive:
    def __init__(self):
        self.n = 0
        self.m = 0
        self.matrix = None

    1 usage
    @staticmethod
    def check_validity(n, m):
        """Checks validity of input and returns bool"""
        try:
            n = int(n)
            m = int(m)
            if n <= 0 or m <= 0:
                return False
            else:
                return True
        except ValueError:
            return False

    2 usages
    def input_values(self):
        """Inputs values of matrix and generates new matrix"""
        print("Enter n and m matrix dimensions.")
        n = input("n: ")
        m = input("m: ")
```



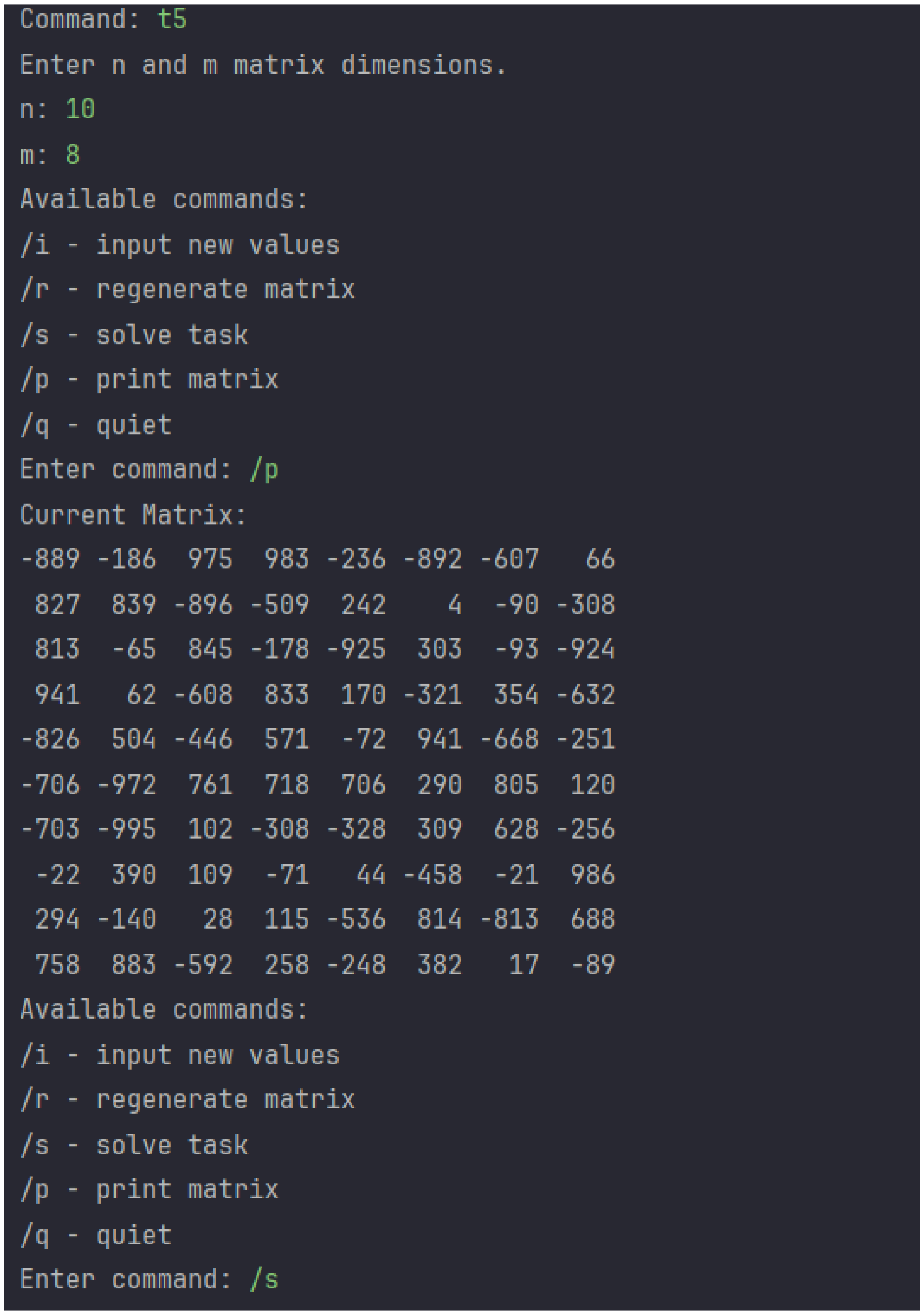
```python
        if TaskFive.check_validity(n, m):
            self.n = int(n)
            self.m = int(m)
        else:
            print("Invalid input.")
        self.generate_matrix()

    2 usages
    def generate_matrix(self):
        """Generates matrix with n rows and m columns of random ints"""
        self.matrix = np.random.randint(-1000, 1000, size=(self.n, self.m))

    1 usage
    def print_matrix(self):
        """Prints matrix"""
        print("Current Matrix:")
        df = pd.DataFrame(self.matrix)
        print(df.to_string(index=False, header=False))

    1 usage
    def solve_personal_task(self):
        """Solve personal(counts even|odd elements of matrix and calculates correlation if possible) task and prints
        results."""
        even_nums = []
        odd_nums = []

        for i in range(self.n):
            for j in range(self.m):
                if self.matrix[i][j] % 2 == 0:
                    even_nums.append(self.matrix[i][j])
```



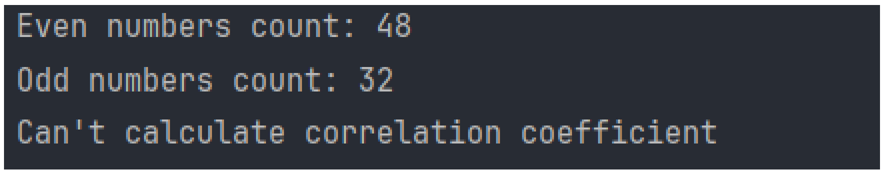
```python
                else:
                    odd_nums.append(self.matrix[i][j])

        # 1
        print(f"Even numbers count: {len(even_nums)}\nOdd numbers count: {len(odd_nums)}")
        # 2
        try:
            if 0 < len(even_nums) == len(odd_nums):
                cor = np.corrcoef(even_nums, odd_nums)[0, 1]
                print(f"Correlation coefficient x/y : {cor}")
            else:
                print("Can't calculate correlation coefficient")
        except Exception as e:
            print(e)

    1 usage
    @staticmethod
    def print_command_list():
        """Prints command list for task 5"""
        print(
            "Available commands:\n/i - input new values\n/r - regenerate matrix\n/s - solve task\n/p - print "
            "matrix\n/q - quiet")

    1 usage
    def run(self):
        while self.n == 0:
            self.input_values()

```



```python
        while True:
            TaskFive.print_command_list()
            command = input("Enter command: ")
            if command == "/i":
                self.input_values()
            elif command == "/r":
                self.generate_matrix()
            elif command == "/s":
                self.solve_personal_task()
            elif command == "/p":
                self.print_matrix()
            elif command == "/q":
                break

```

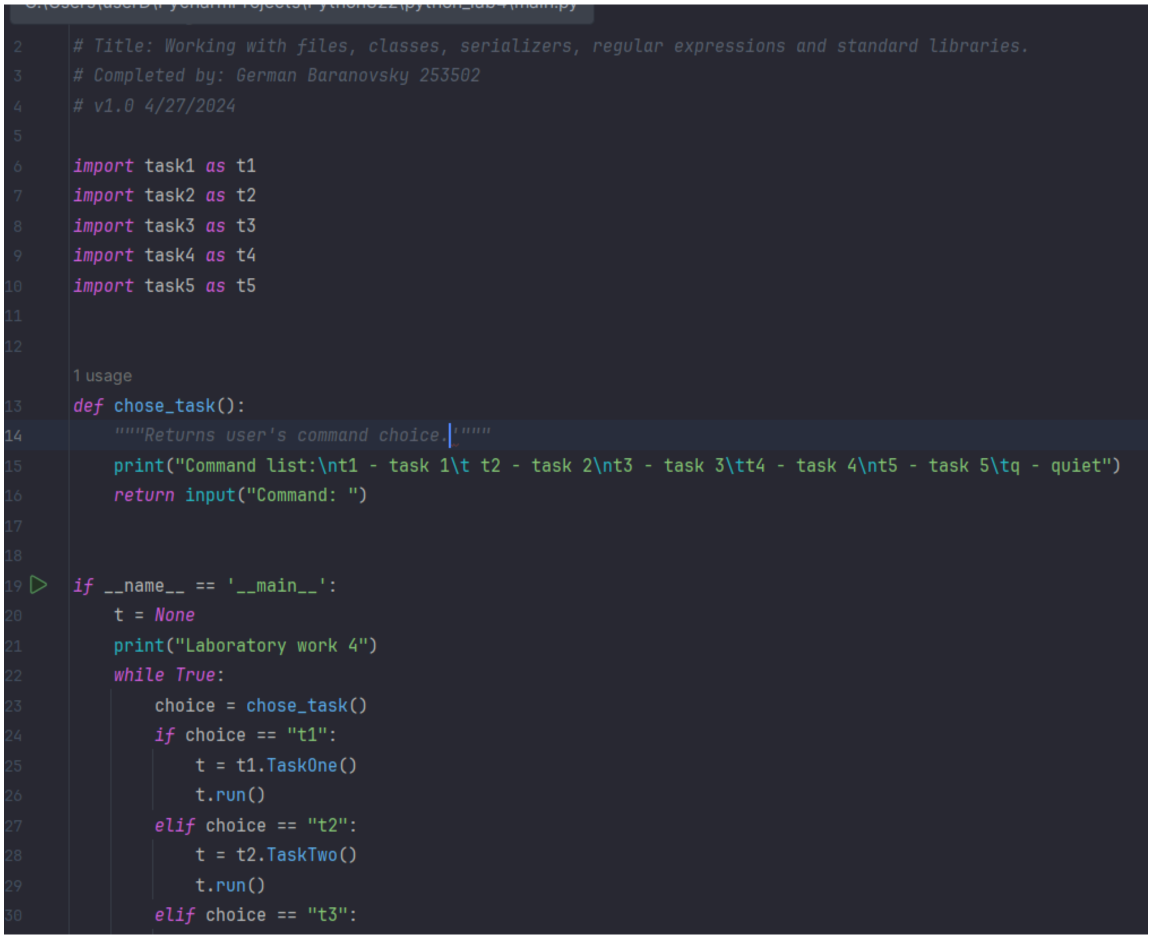


```
Command: t5
Enter n and m matrix dimensions.
n: 10
m: 8
Available commands:
/i - input new values
/r - regenerate matrix
/s - solve task
/p - print matrix
/q - quiet
Enter command: /p
Current Matrix:
-889 -186  975  983 -236 -892 -607   66
 827  839 -896 -509  242    4  -90 -308
 813  -65  845 -178 -925  303  -93 -924
 941   62 -608  833  170 -321  354 -632
-826  504 -446  571  -72  941 -668 -251
-706 -972  761  718  706  290  805  120
-703 -995  102 -308 -328  309  628 -256
 -22  390  109  -71   44 -458  -21  986
 294 -140   28  115 -536  814 -813  688
 758  883 -592  258 -248  382   17  -89
Available commands:
/i - input new values
/r - regenerate matrix
/s - solve task
/p - print matrix
/q - quiet
Enter command: /s
```

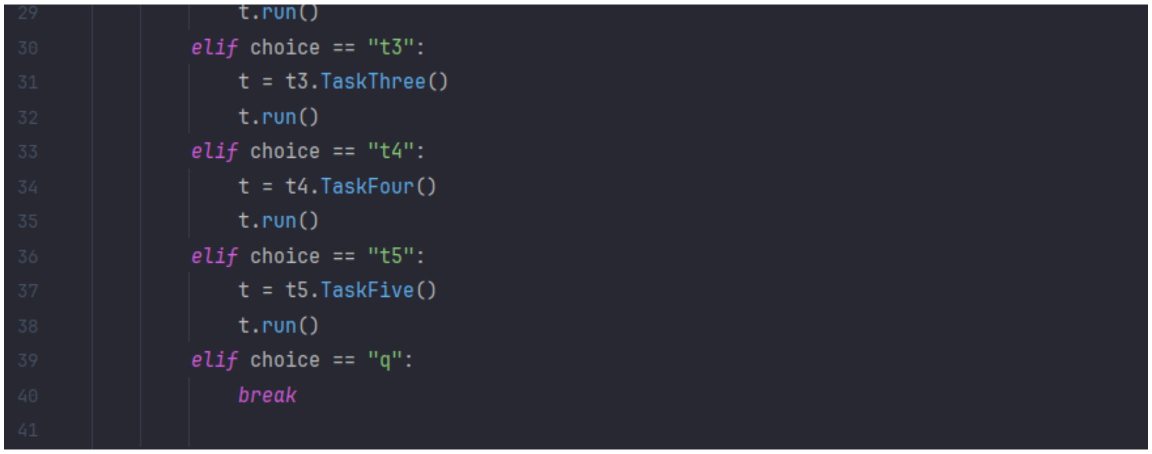


```
Even numbers count: 48
Odd numbers count: 32
Can't calculate correlation coefficient
```

6. Файл main.py



```python
# Title: Working with files, classes, serializers, regular expressions and standard libraries.
# Completed by: German Baranovsky 253502
# v1.0 4/27/2024

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


1 usage
def chose_task():
    """Returns user's command choice."""
    print("Command list:\nt1 - task 1\t t2 - task 2\nt3 - task 3\tt4 - task 4\nt5 - task 5\tq - quiet")
    return input("Command: ")


if __name__ == '__main__':
    t = None
    print("Laboratory work 4")
    while True:
        choice = chose_task()
        if choice == "t1":
            t = t1.TaskOne()
            t.run()
        elif choice == "t2":
            t = t2.TaskTwo()
            t.run()
        elif choice == "t3":
```



```python
                t.run()
            elif choice == "t3":
                t = t3.TaskThree()
                t.run()
            elif choice == "t4":
                t = t4.TaskFour()
                t.run()
            elif choice == "t5":
                t = t5.TaskFive()
                t.run()
            elif choice == "q":
                break
```