МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный морской технический университет»  
 (СПБГМТУ)

|  |
| --- |
| ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ  КАФЕДРА КИБЕРФИЗИЧЕСКИХ СИСТЕМ |

КУРСОВОЙ ПРОЕКТ

по дисциплине “программирование”

|  |
| --- |
|  |

|  |
| --- |
| Выполнила студентка группы 20121 |
| Плетнева Ева Анатольевна |
| Проверил |
| Поделенюк П. П. |

Санкт-Петербург

2023 год.

Оглавление

[Цели 3](#_Toc150114211)

[Формулировка задачи 4](#_Toc150114212)

[Разработка классов и UML диаграмма 7](#_Toc150114213)

[Результаты работы 9](#_Toc150114214)

[Заключение 24](#_Toc150114215)

[Список используемых источников 25](#_Toc150114216)

# Цели

**Цель:** Анализ ходов фигуры на шахматной доске.

При работе над данным лабораторным проектом будут затронуты следующие

вопросы:

1. Работа с основами функционального программирования языка Python

2. Работы с основами ООП языка Python

3. Разработка классов и UML диаграмма

4. Работы с пакетами Python

5. Создание GUI приложения с использованием tkinter (или customtkinter)

+ pygame (+OpenGl).

# Формулировка задачи

**Формулировка задания**: Дана квадратная шахматная доска размером N x N. На доске уже размещено K фигур. Фигуры размещены так, что находятся не под боем друг друга. Необходимо расставить на доске еще L фигур так, чтобы никакая из фигур на доске не находилась под боем любой другой фигуры. Требуется найти одно решение для визуализации и все возможные решения для вывода в файл. Если решение не найдено, то необходимо вывести соответствующее сообщение.

1) Необходимо создать UML диаграмму взаимодействия классов. При необходимости можно делать несколько UML диаграмм. Для диаграммы создается табличка взаимодействия (Имя класса :: Имя базового класса :: Описание). Также для каждого класса создается таблица методов и атрибутов (Методы и атрибуты :: Описание ). При описании методов обязательно прописывать тип параметров, а также выходной тип данных. При описании атрибутов необходимо прописывать его тип.

2) Необходимо составить программу с использованием функционального программирования языка Python, где:

1. Входные данные в файле input.txt. На первой строке файла записаны три числа: N L K (через пробел). Далее следует K строк, содержащих числа x и y (через пробел) - координаты уже стоящей на доске фигуры (фигуры стоят правильно). Координаты отсчитываются от 0 до N-1. 1 <= N <= 20.

2. Выходные данные в файл output.txt. На каждое найденное решение необходимо записать в файл одну строку. Строка состоит из пар (x,y) - координаты фигур на доске. В решение следует вывести координаты всех фигур, находящихся на доске. Каждую фигуру необходимо записать в виде пары координат, разделенных запятой и обрамленных скобками. Координаты отсчитываются от 0 до N- 1. Порядок, в котором фигуры перечислены в решении, не имеет значения. Если не было найдено ни одного решения, в файл необходимо записать no solutions.

3. Выходные данные на консоль — это доска N\*N, где фигура обозначается #, ее ходы обозначаются \*, а пустые клетки обозначаются 0.

3) Необходимо составить программу с использованием ООП языка Python, а также разработать интерфейса при помощи пакетов tkinter (или customtkinter) + pygame (+OpenGl). Задача - реализовать предложенный вариант:

2. tkinter (или customtkinter) + pygame(+OpenGl)).

2.1.tkinter (или customtkinter)

2.1.1. На интерфейсе есть три поля ввода и кнопка. В первом поле вводится размер доски N, во втором – количество фигур, которые необходимо расставить с помощью алгоритма L, в третьем – количество стоящих на доске фигур K. С помощью кнопки создается новое окно. Необходимо проверять правильность ввода данных.

2.1.2. В данном интерфейсе есть K полей ввода и кнопка. В каждое поле необходимо ввести числа x и y (через пробел) - координаты уже стоящей на доске фигуры. С помощью кнопки создается новое окно. Необходимо проверять правильность ввода данных.

2.2. pygame (+OpenGl)

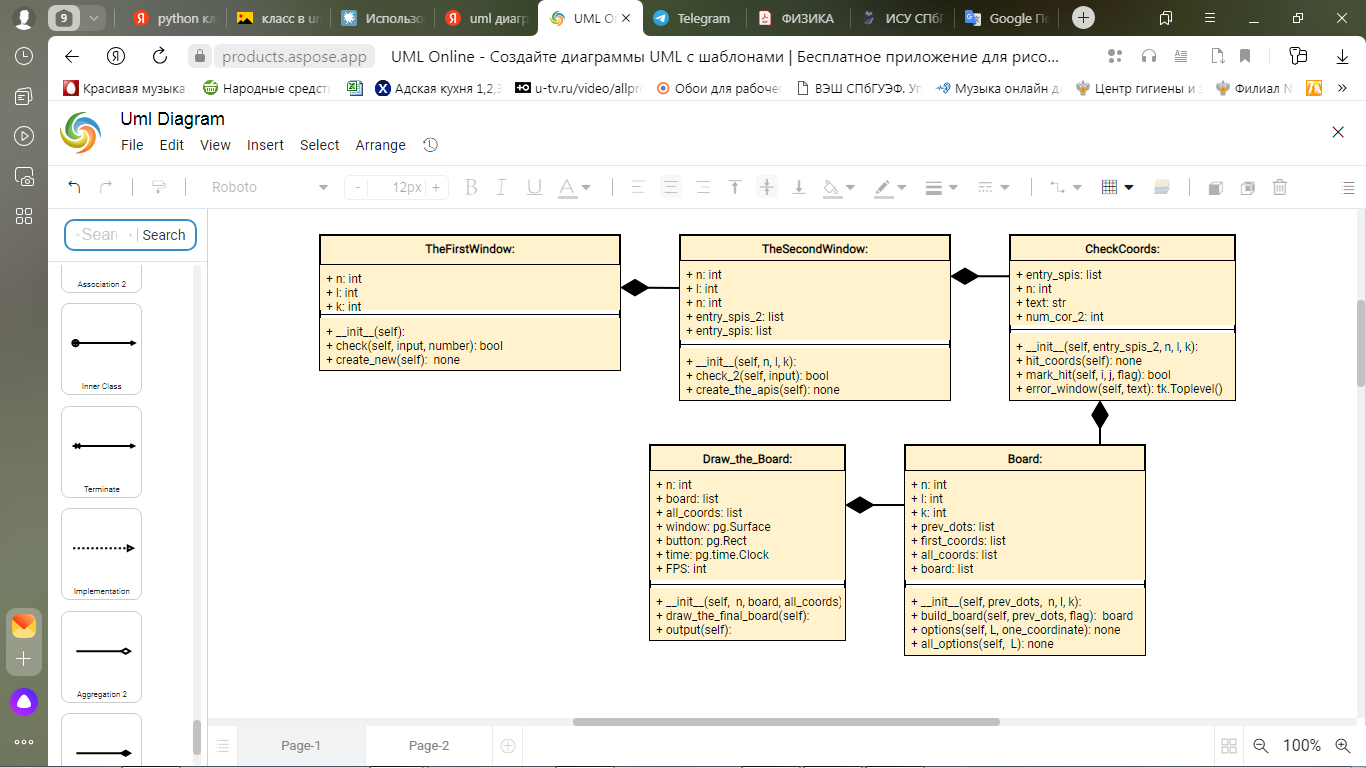
2.2.1. В данном интерфейсе есть шахматная доска и кнопка. На созданной шахматной доске показываются фигуры, расставленные пользователем и найденные алгоритмом, (нужно показать любое найденное решение). Визуализация пользовательских и найденных алгоритмом фигур должна отличаться (визуализация ходов одинаковая). Если решение не было найдено, то вывести сообщение об этом и закрыть данный интерфейс. С помощью кнопки происходит вывод данных в файл output.txt. На каждое найденное решение необходимо записать в файл одну строку. Строка состоит из пар (x,y) - координаты фигур на доске. В решение следует вывести координаты всех фигур, находящихся на доске. Каждую фигуру необходимо записать в виде пары координат, разделенных запятой и обрамленных скобками. Координаты отсчитываются от 0 до N. Порядок, в котором фигуры перечислены в решении, не имеет значения.

*Дополнительная информация*:

1. Обязательное использование аннотации и комментариев

2. Проверка ввода данных.

# Разработка классов и UML диаграмма



|  |  |
| --- | --- |
| Классы | Методы класса |
| **TheFirstWindow**  Класс, создающий окно для ввода N, L, K и проверяющий правильность введенных данных | **check** - Метод для проверки корректности введенных данных  **create\_new**- Метод кнопки для вызова окна для введения координат |
| **TheSecondWindow**  Класс, создающий окно для введения координат | **сheck\_2** - Метод, проверяющий правильность введенных значений  **сreate\_the\_apis** -Метод, вызывающий класс для проверки координат |
| **CheckCoords**  Класс для проверки на совпадение координат и расположение фигур под боем друг у друга | **hit\_coords** - Метод для перебора координат фигур и создания пояснения к ошибке, если таковая имеется  **mark\_hit** - Метод, проверяющий, находится ли фигура под боем другой, или расставляющий на доске места, куда фигура может походить  **error\_window** - Метод для создания окна с пояснением к ошибке |
| **Board**  Класс для поиска всех вариантов расстановки фигур и создания матрицы в виде доски для дальнейшей ее отрисовки в pygame" | **build\_board** - Метод создает доску для поиска новых решений или для ее отрисовки  options - Метод, продолжающий поиск мест для постановки фигур и записывающий в массив все варианты координат фигур на доске  **all\_options** - Метод, начинающий поиск мест для постановки фигур |
| **Draw\_the\_Board**  Класс для создания окна в pygame с графическим представлением доски и кнопки для записи всех решений в файл | **draw\_the\_final\_board** - Метод для отрисовки доски в pygame и отслеживания события нажатия на кнопку 'Записать'  output - Метод для записи всех вариантов расстановки фигур в файл |

# Результаты работы

1. Реализация программы с использованием функционального программирования языка Python.
   1. **Ход работы.**

Функции программы:

**mark\_hit** – сперва находит координаты, куда может походить фигура на заданной доске, и проставляет «\*» вместо символов «0», координаты которых соответствуют найденным.

**build\_board** –строит доску в виде матрицы, заполняя ее «0», отмечает на ней фигуры в виде «#», затем, использует функцию mark\_hit для отметки мест возможного хода фигур.

**all\_options** – рекурсивная функция, которая находит через циклы for свободные места для расстановки фигур (ими являются элементы «0») и меняет его на «#» на доске. Затем для каждой найденной координаты такого места вызывается функция options. Как только им были найдены все варианты расстановок оставшихся фигур, функция обратно заменяет условный символ фигуры обратно на «#». Если вариантов не было найдено, то выводит «no solutions».

**options** – рекурсивная функция, продолжающая поиск вариантов расстановки фигур и записывающая их в файл. Ведет поиск по аналогии с функцией «**all\_options»**.Как только нужно кол-во фигур было расставлено, о чем будет свидетельствовать переменная L, значение которой уменьшается на единицу с каждым новым вызовом, вариант расстановок фигур в виде списка координат записывается в файл. Также на этапе записи данных в файл «output» метод поменяет значение переменной no\_solutions с True на False. Эта переменная нужна для того, чтобы дать понять предыдущему методу, что ни одного решения не было найдено.

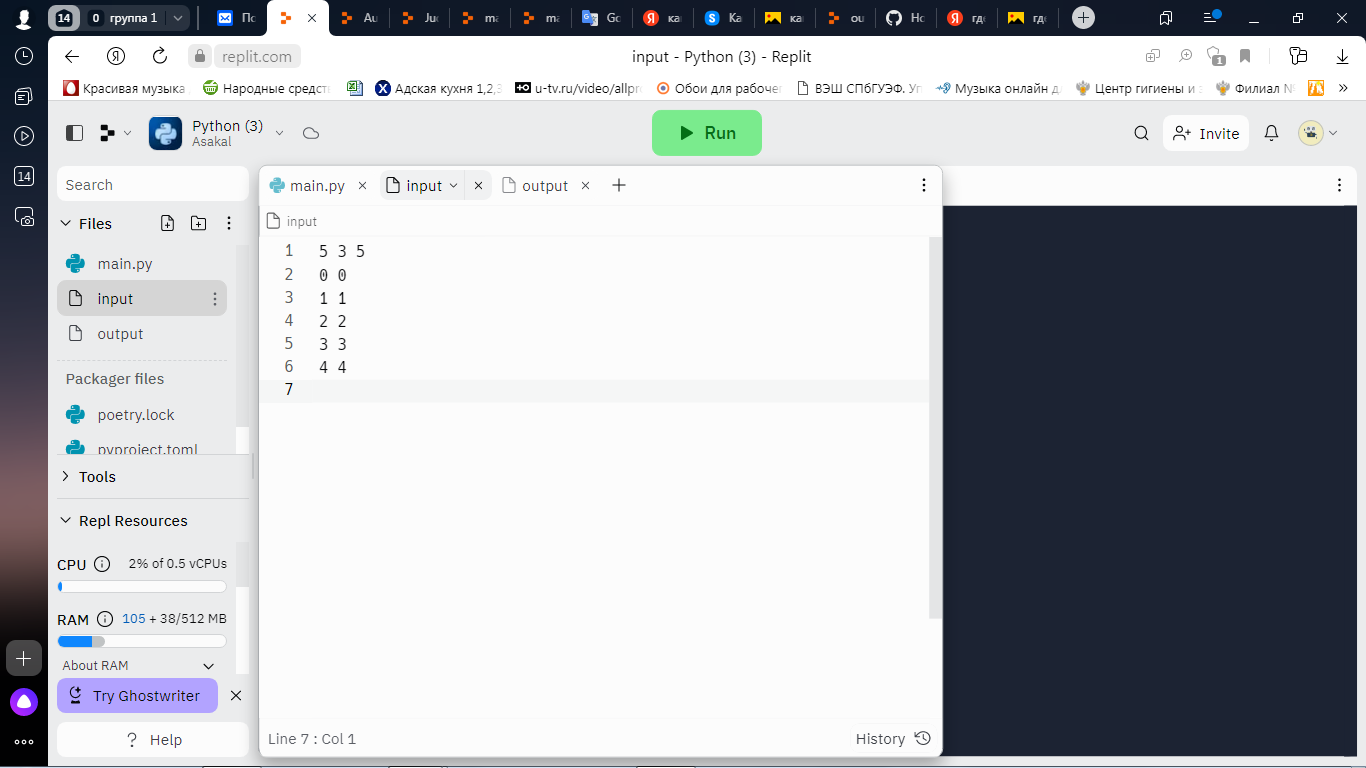
**print\_board** - красиво выводит на печать доску через цикл for.

Сначала считываются данные с файла input, присваиваются эти данные переменным.

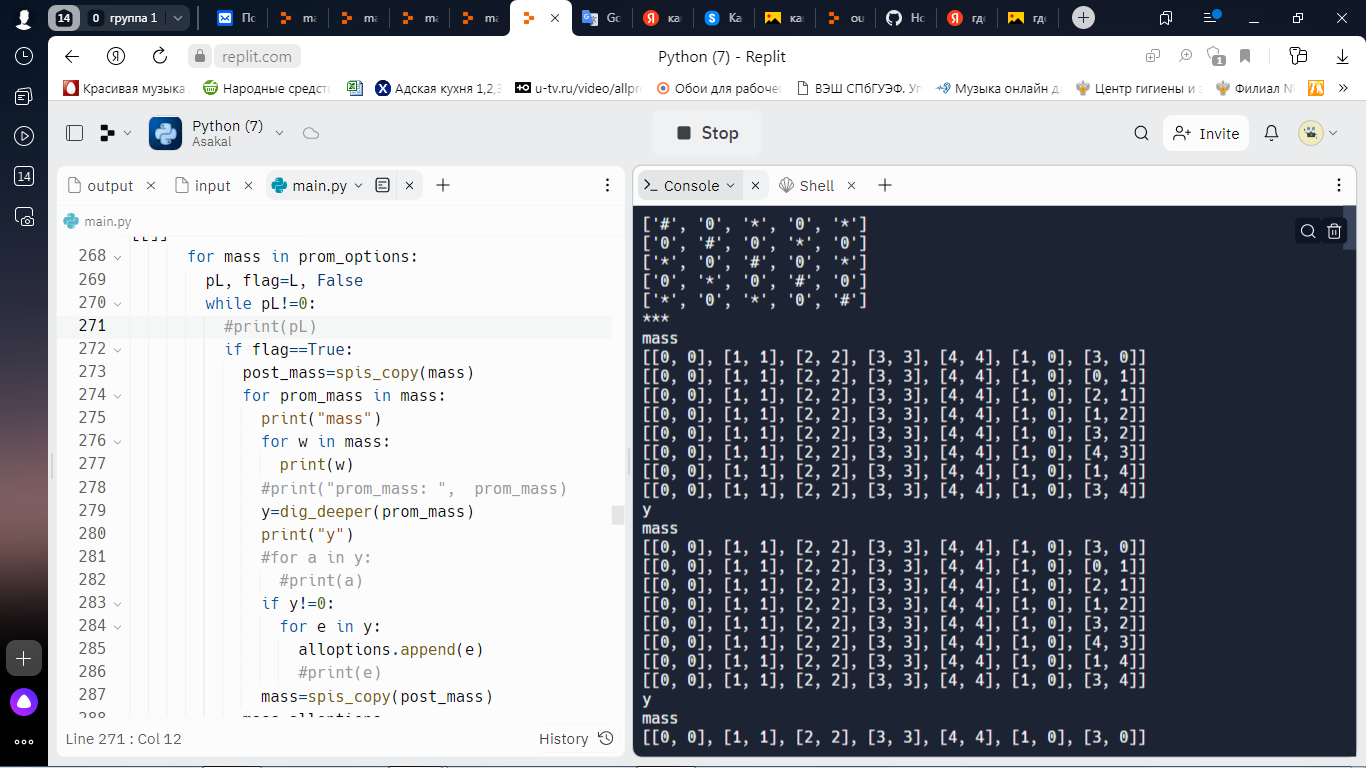
Строится доска, расставляются на нее фигуры и отмечаются места, куда нельзя ставить новые фигуры. Эта доска выводится на консоль. Вызывается функция «**all\_options**», и все списки координат записываются в файл «output».

* 1. **Демонстрация работы программы.**

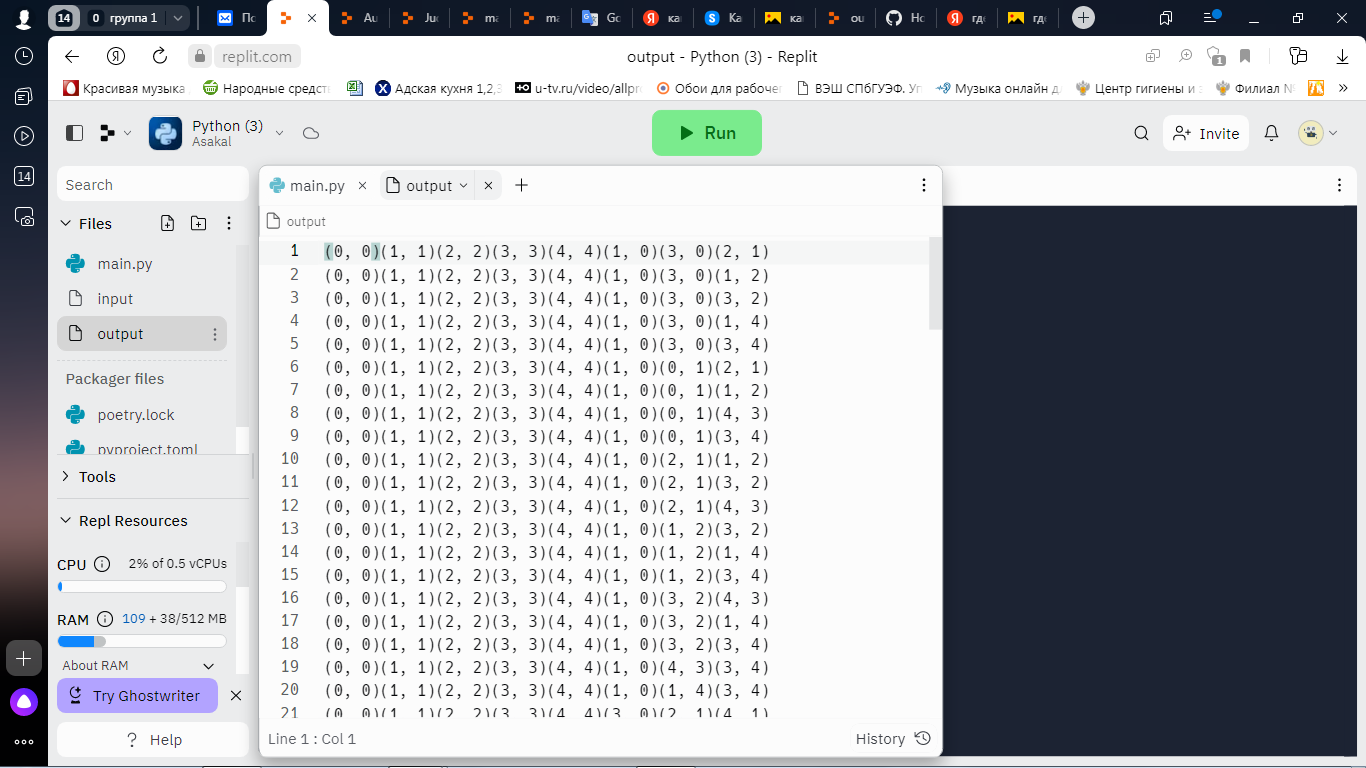
Входные данные.

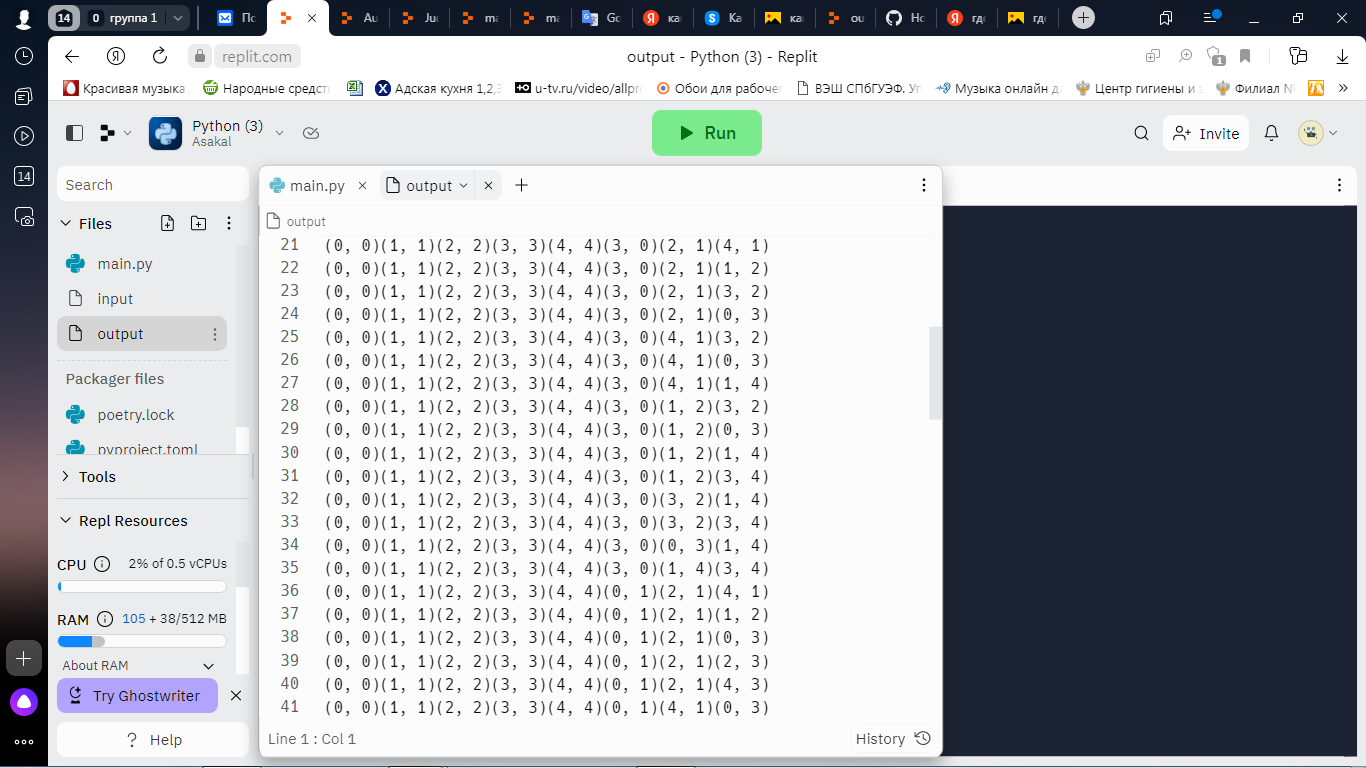


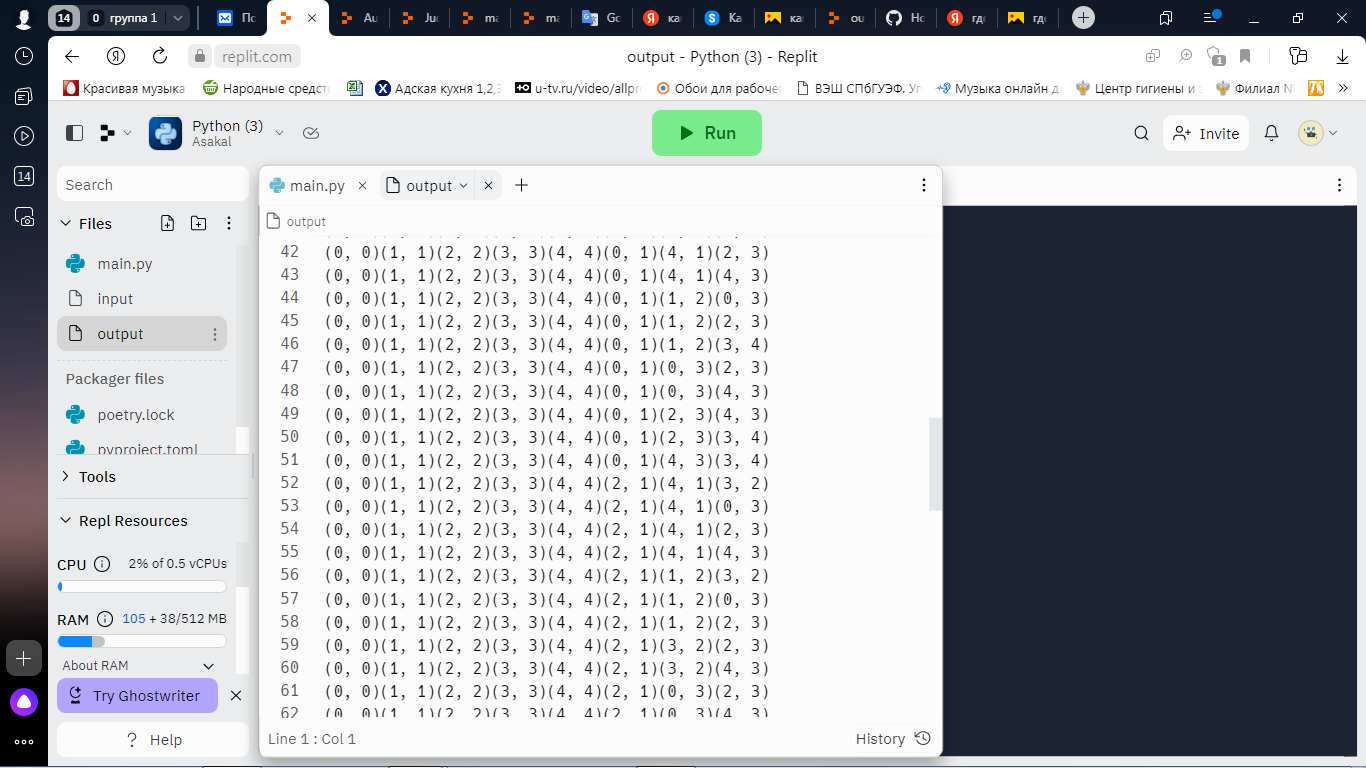
Выходные данные в консоль:

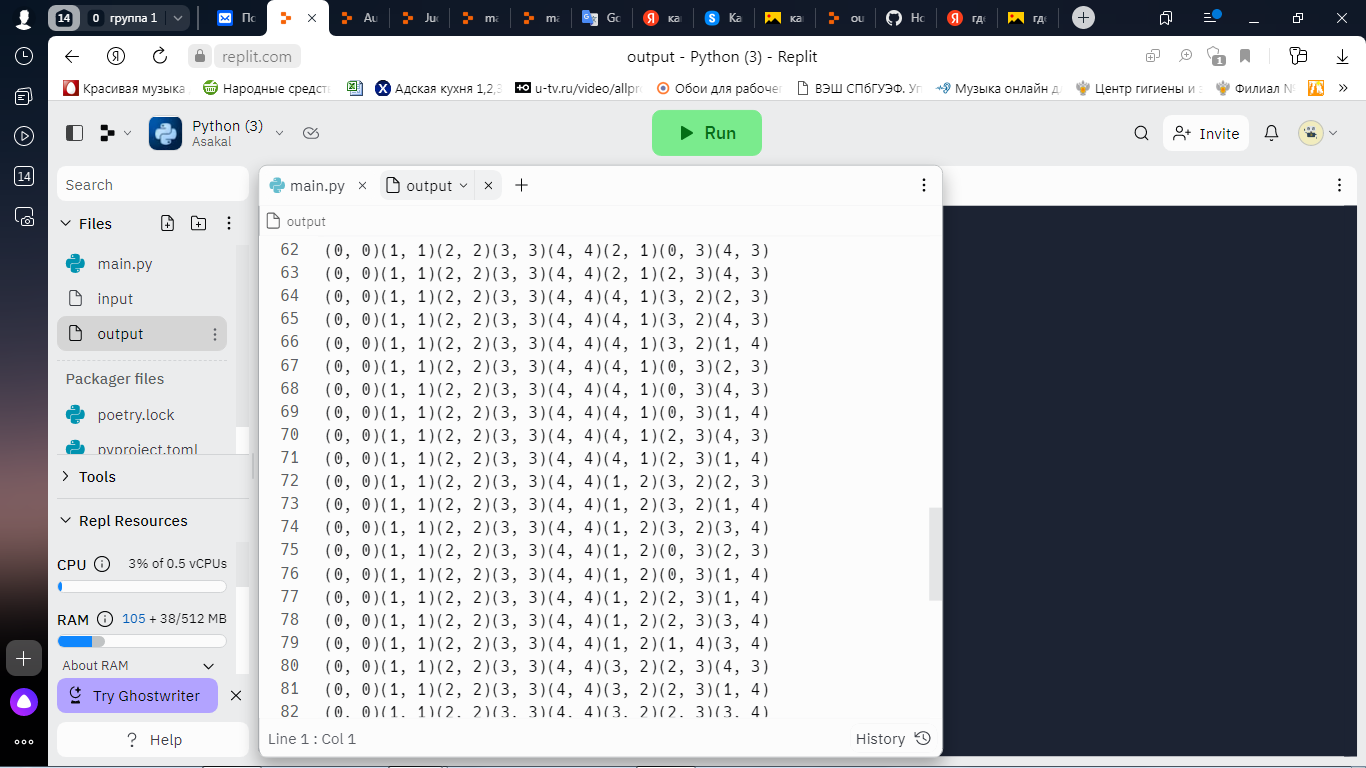


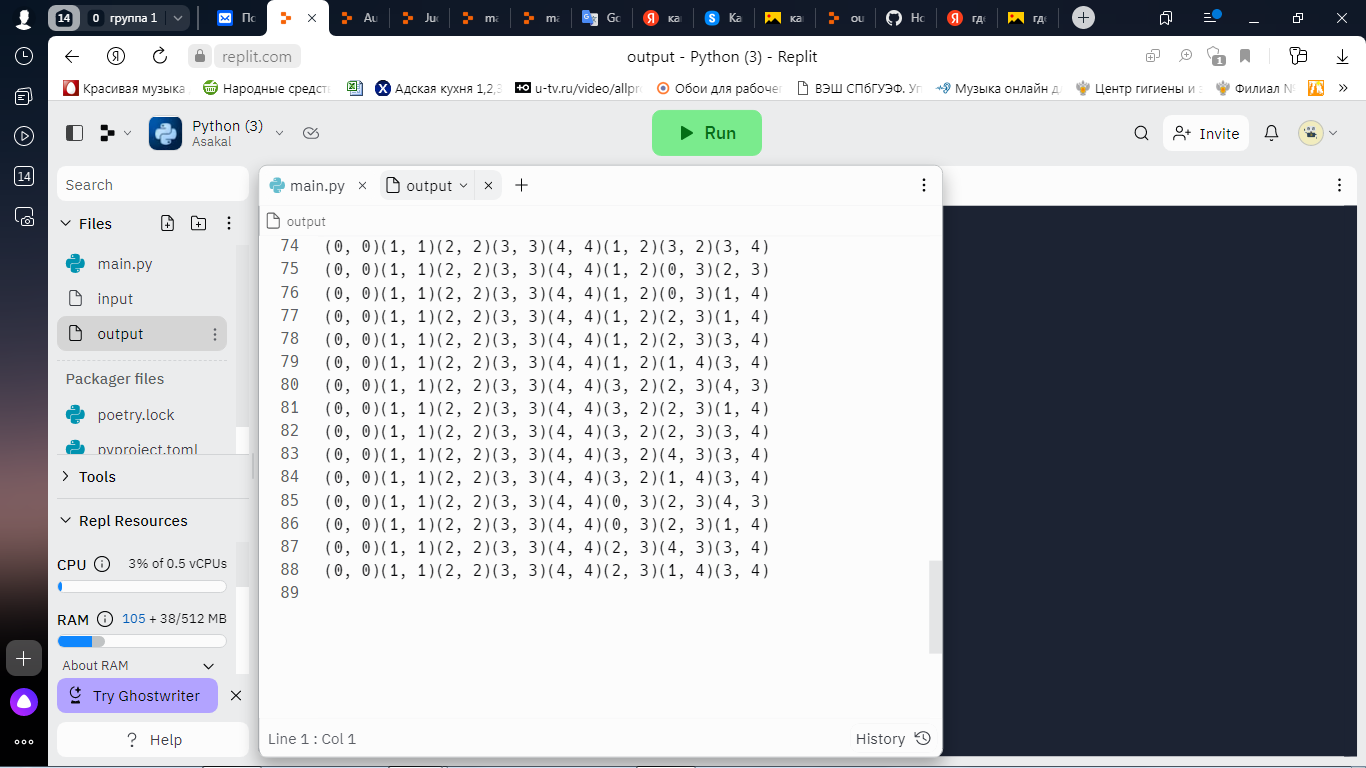
Выходные данные в файл:



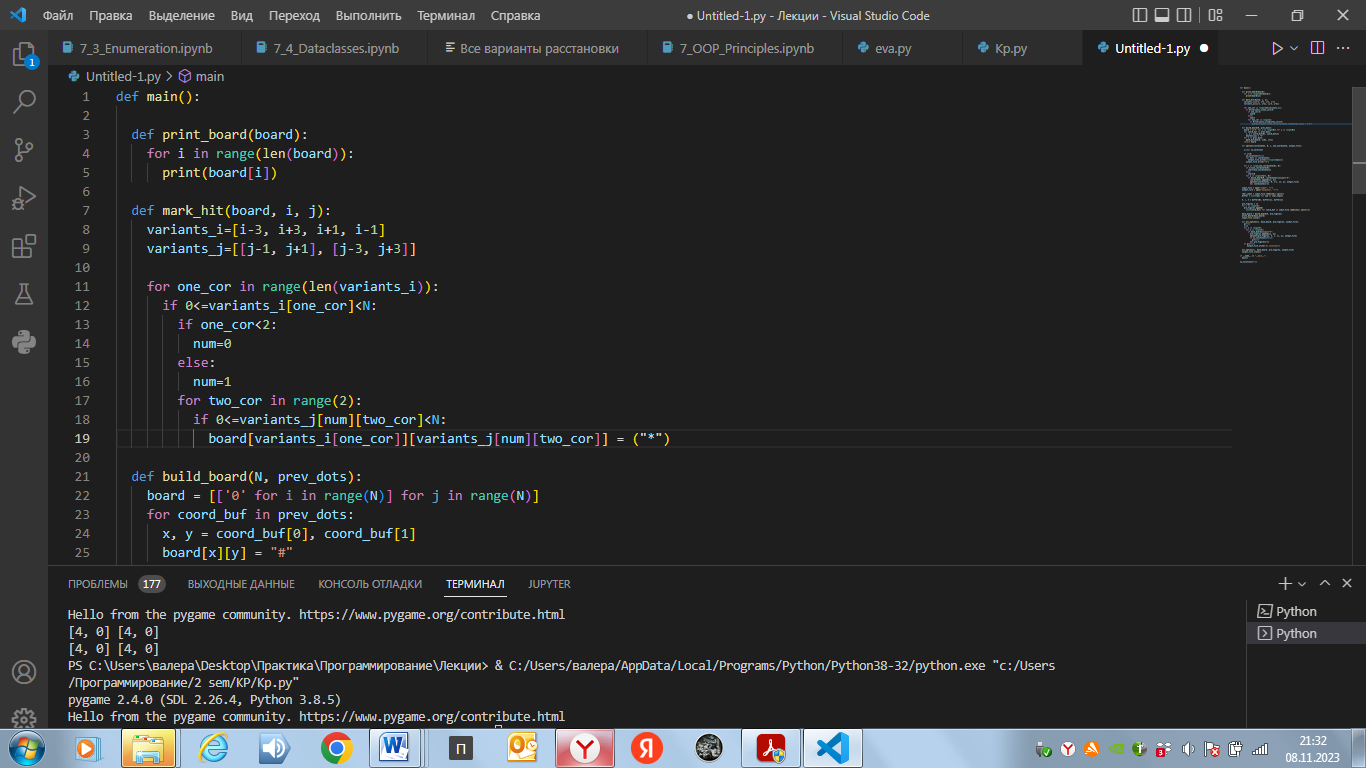


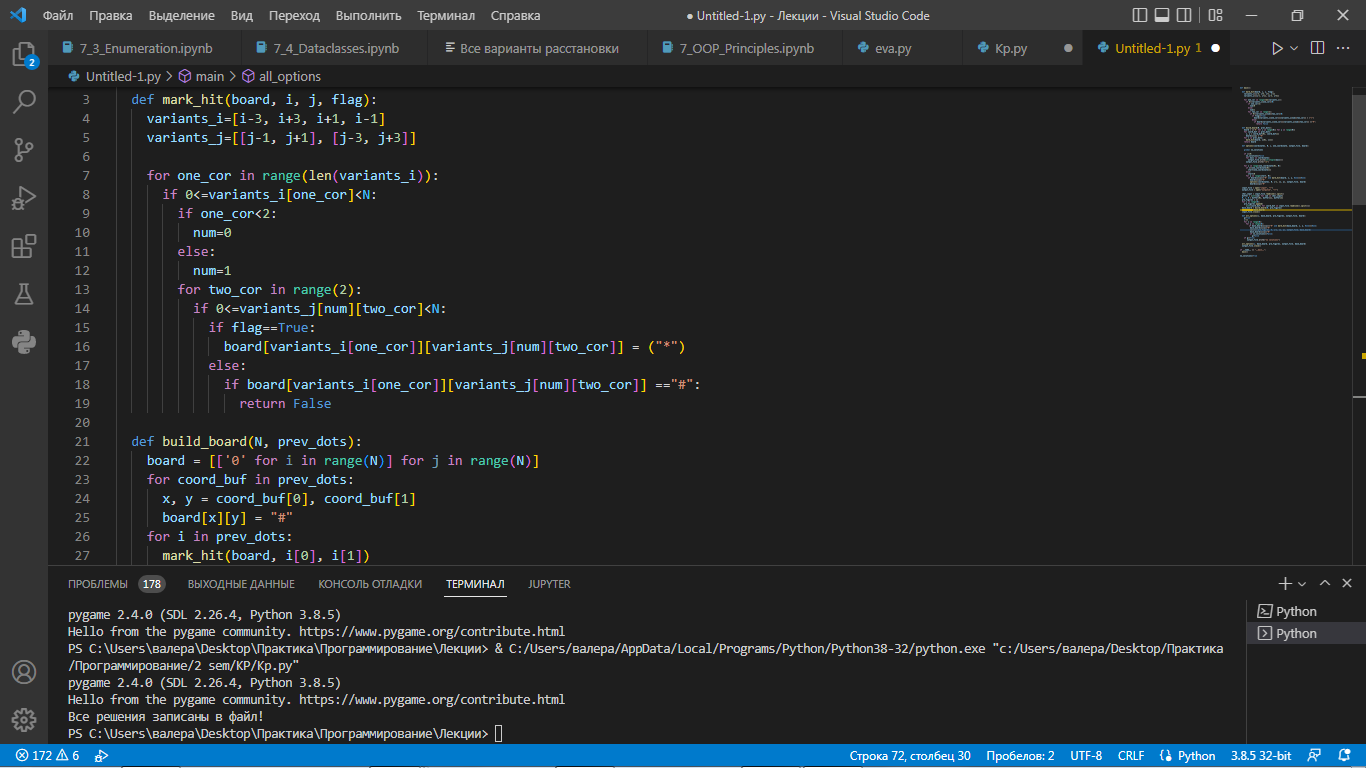


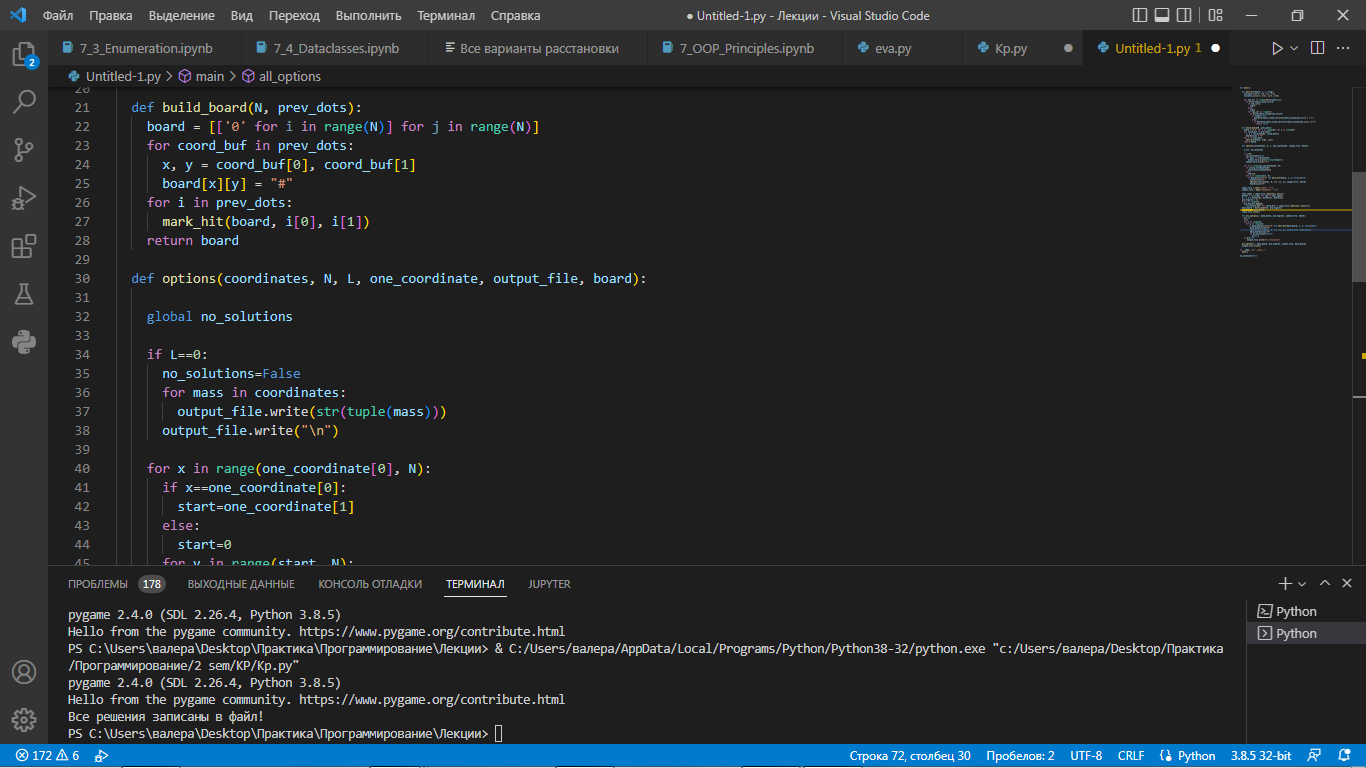


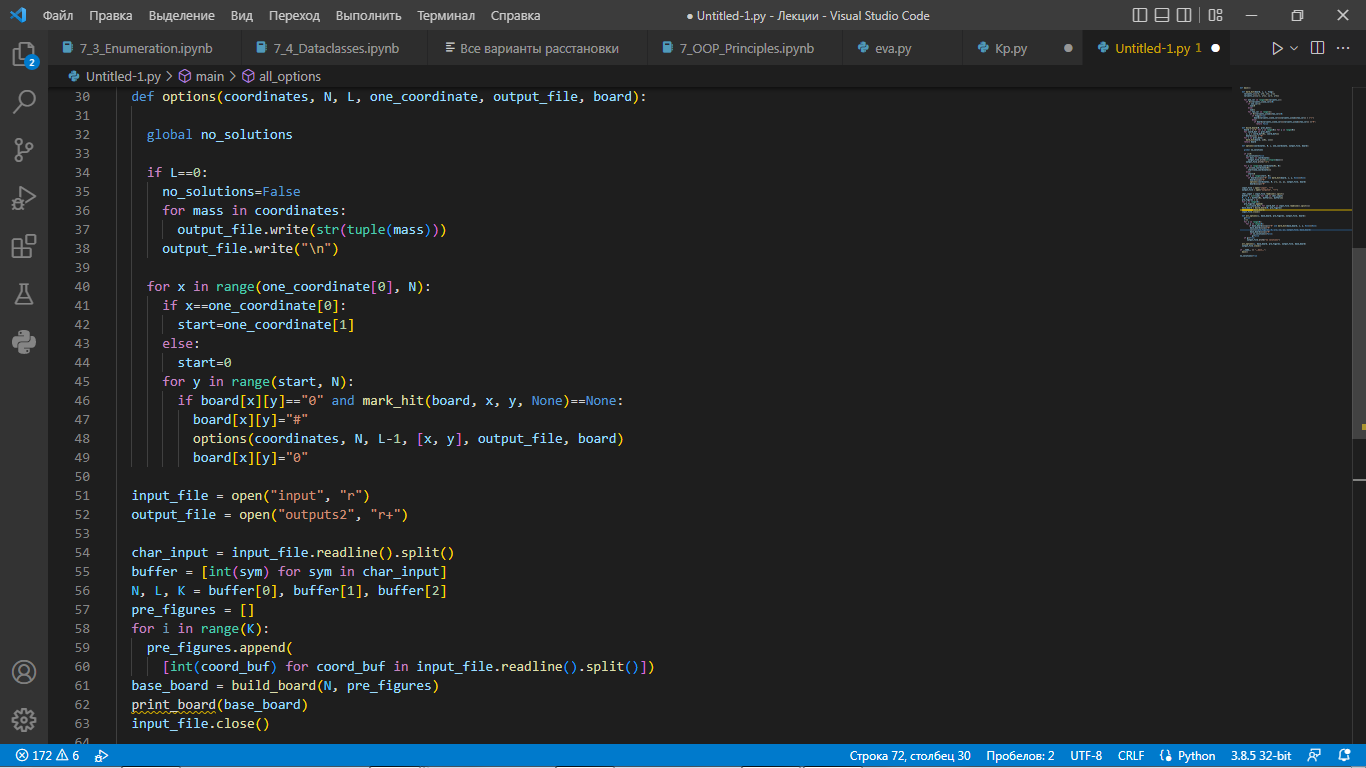


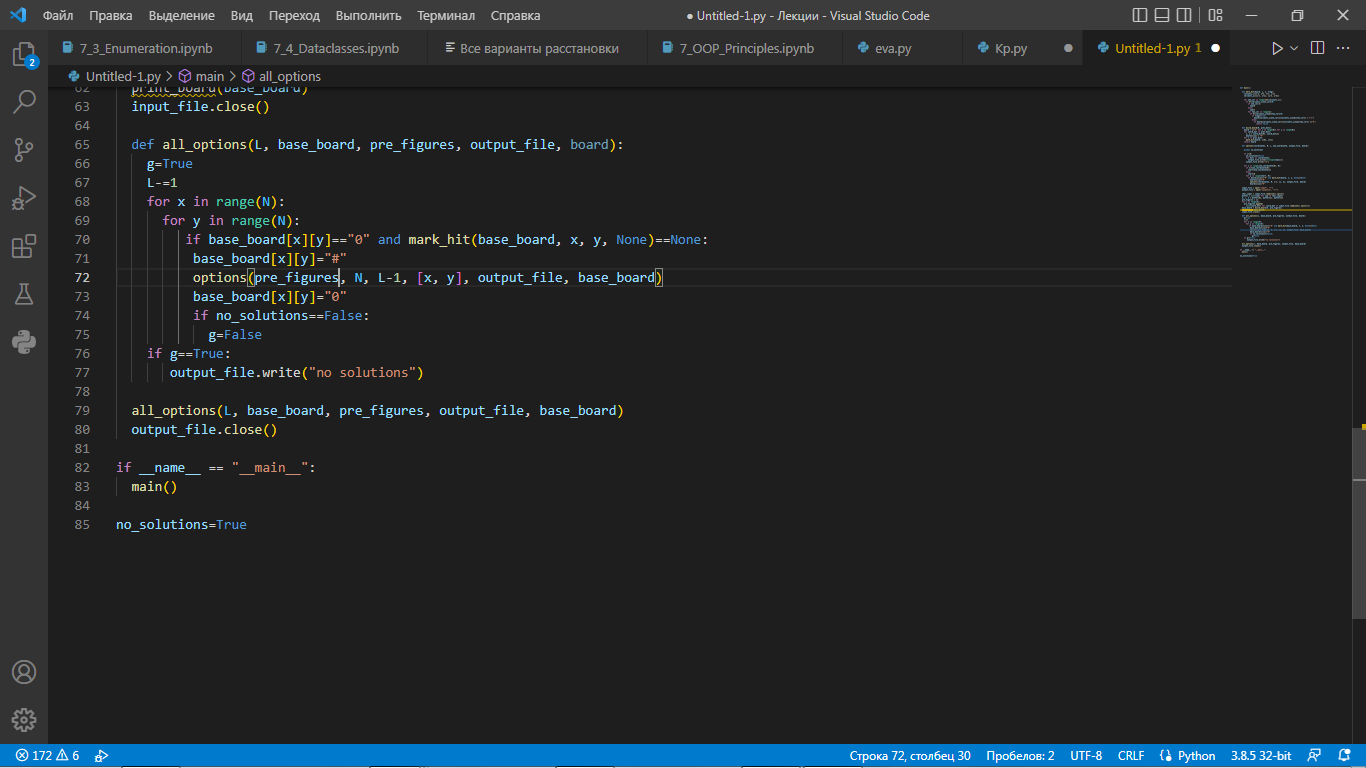
* 1. **Листинг кода**











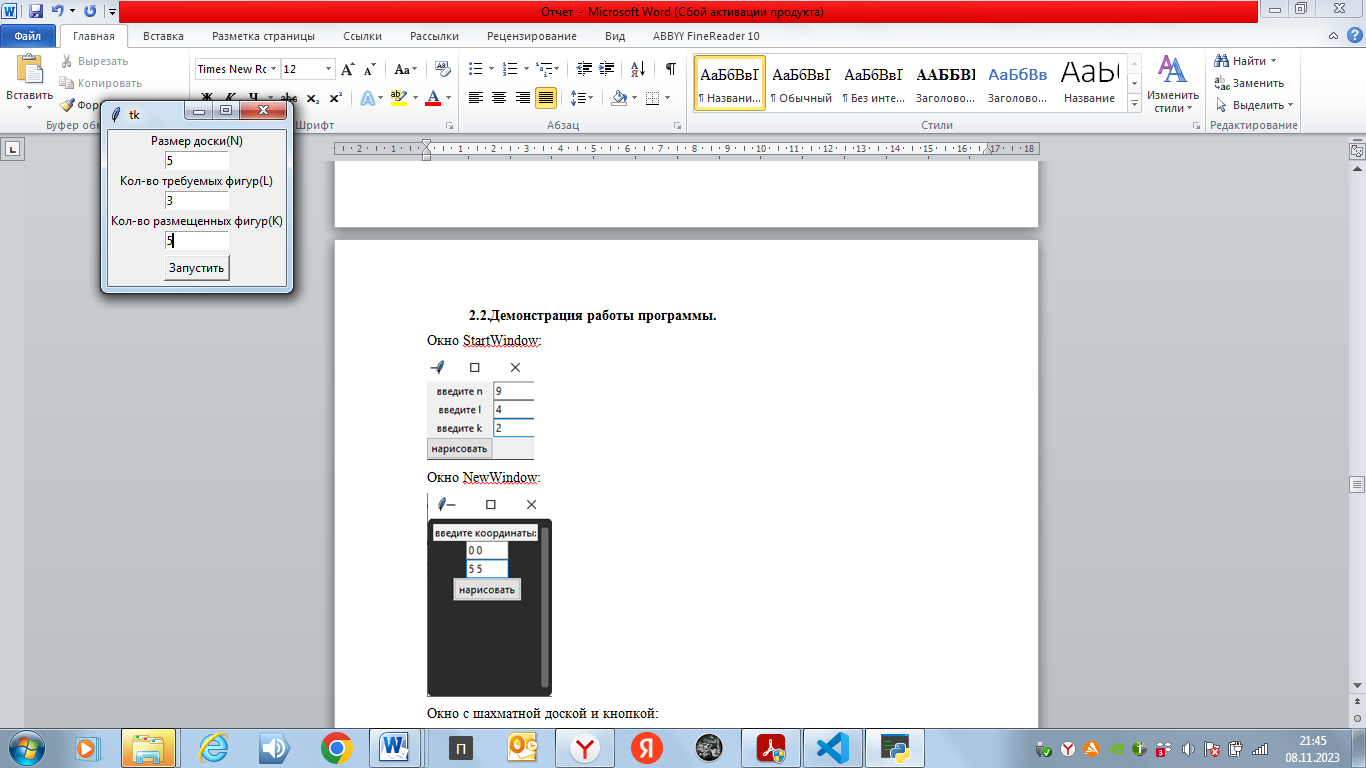
1. **Реализация программы с использованием ООП языка Python.**
   1. **Ход работы.**

При запуске программы сначала срабатывает класс **TheFirstWindow**, создающий окно для ввода размера доски, кол-ва необходимых фигур и кол-во введенных пользователем фигур. При вводе данных методом check при помощи функции validate проверяется их корректность(не вводит ли пользователь буквенные и другие нецелочисленные символы, не превышает ли введенное число определенного значения). Снизу расположена кнопка, которая запускает класс **TheSecondWindow**. Этот класс выводит на экран окно для ввода координат уже поставленных фигур. Если фигур больше трех, то создается колесо прокрутки при помощи модуля «customtkinter». Снизу также присутствует кнопка, запускающая класс **CheckCoords**. Он ведет проверку, не являются ли данные от пользователя одинаковыми и не попадает ли какая-либо фигура под бой другой. Если была найдена ошибка, создается окно с текстом, поясняющим эту ошибку. В ином случае идет вызов класса **Board**, использующий алгоритм для поиска всех вариантов расстановок фигур. Если ни один не был найден вызывается метод «error\_window» из **CheckCoords** для вывода окна с ошибкой, иначе запускается класс **Draw**, запускающий pygame с доской в виде цветных клеток и кнопкой для записи в файл всех решений. Каждый цвет клетки имеет свое значение:

*бордовый* – клетка, где стоит фигура с заданным пользователем местоположением; *зеленый* - клетка, где стоит фигура с местоположением, найденным алгоритмом; *белый* – место, куда может походить одна из фигур, *коричневый* – свободное место.

* 1. **Демонстрация работы программы.**

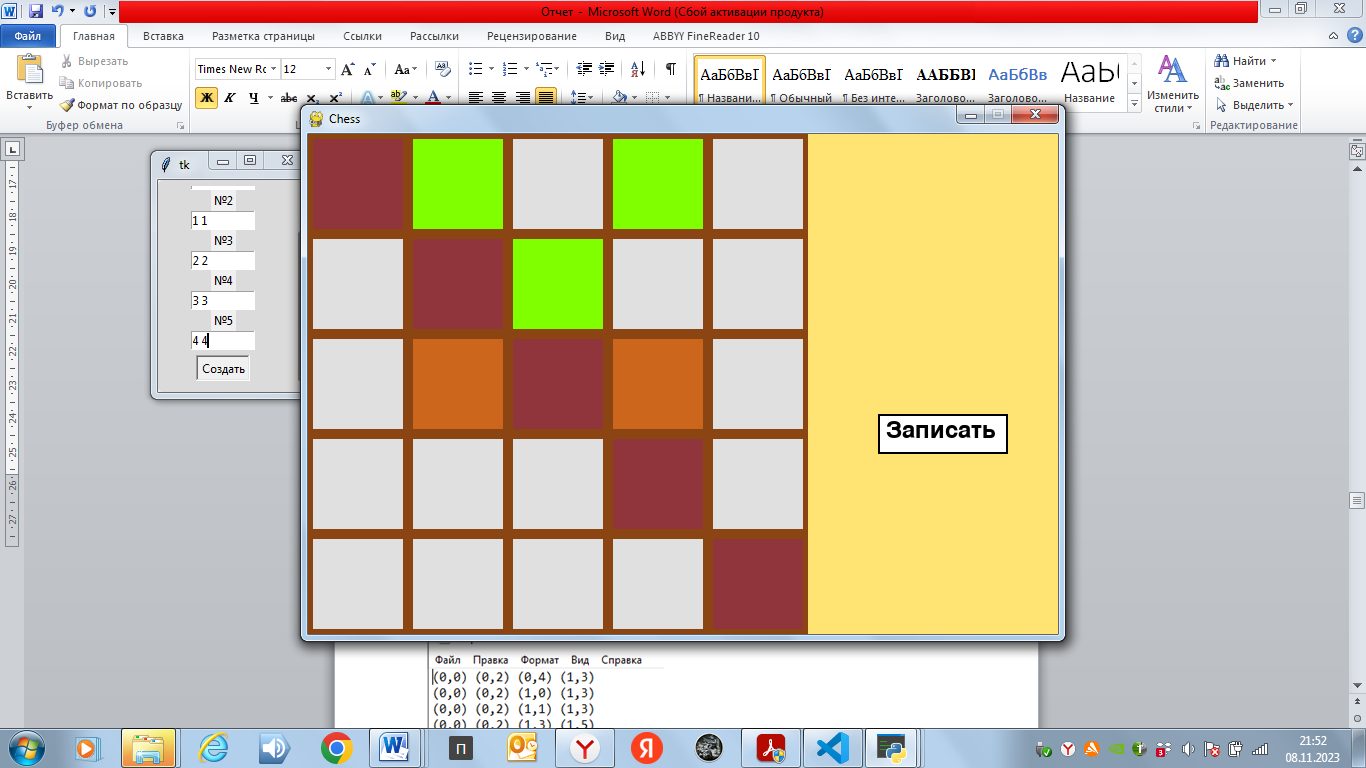
Окно TheFirstWindow:



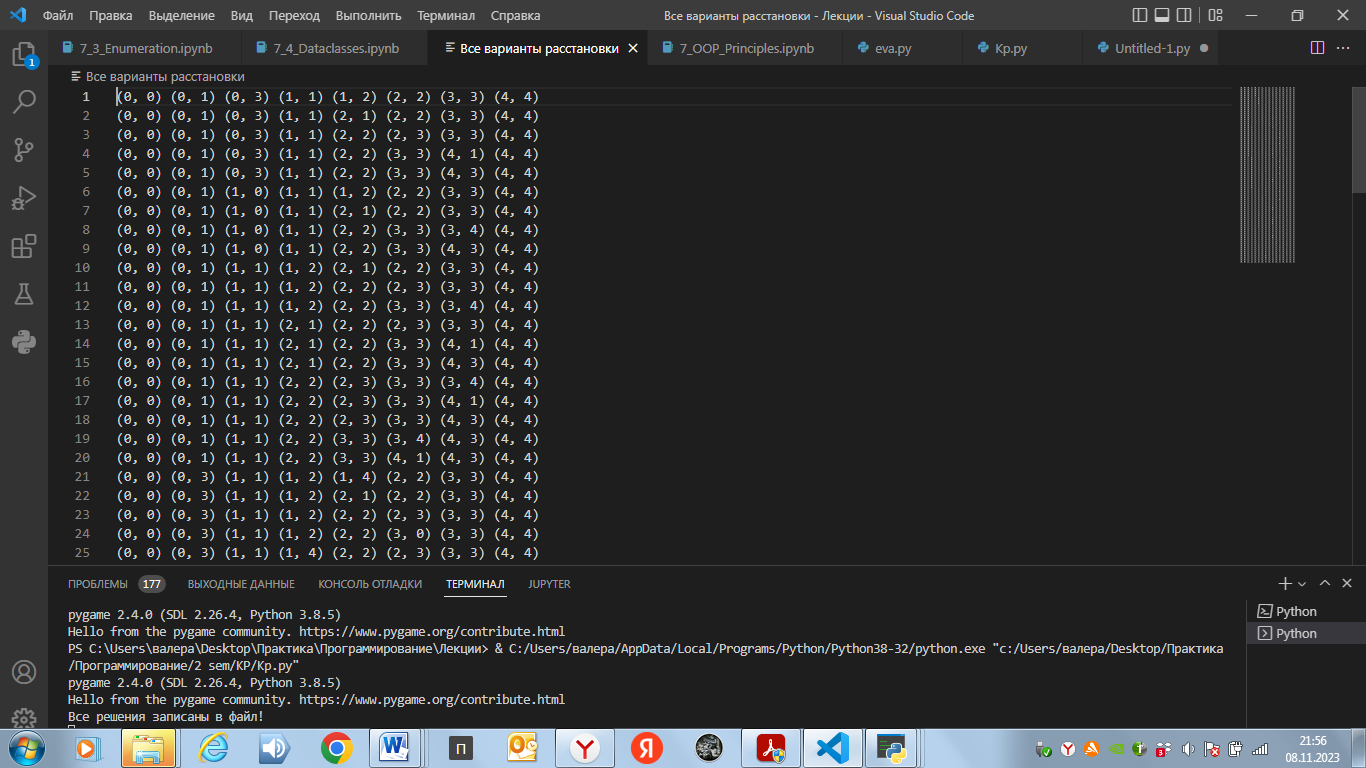
Окно TheSecondWindow:

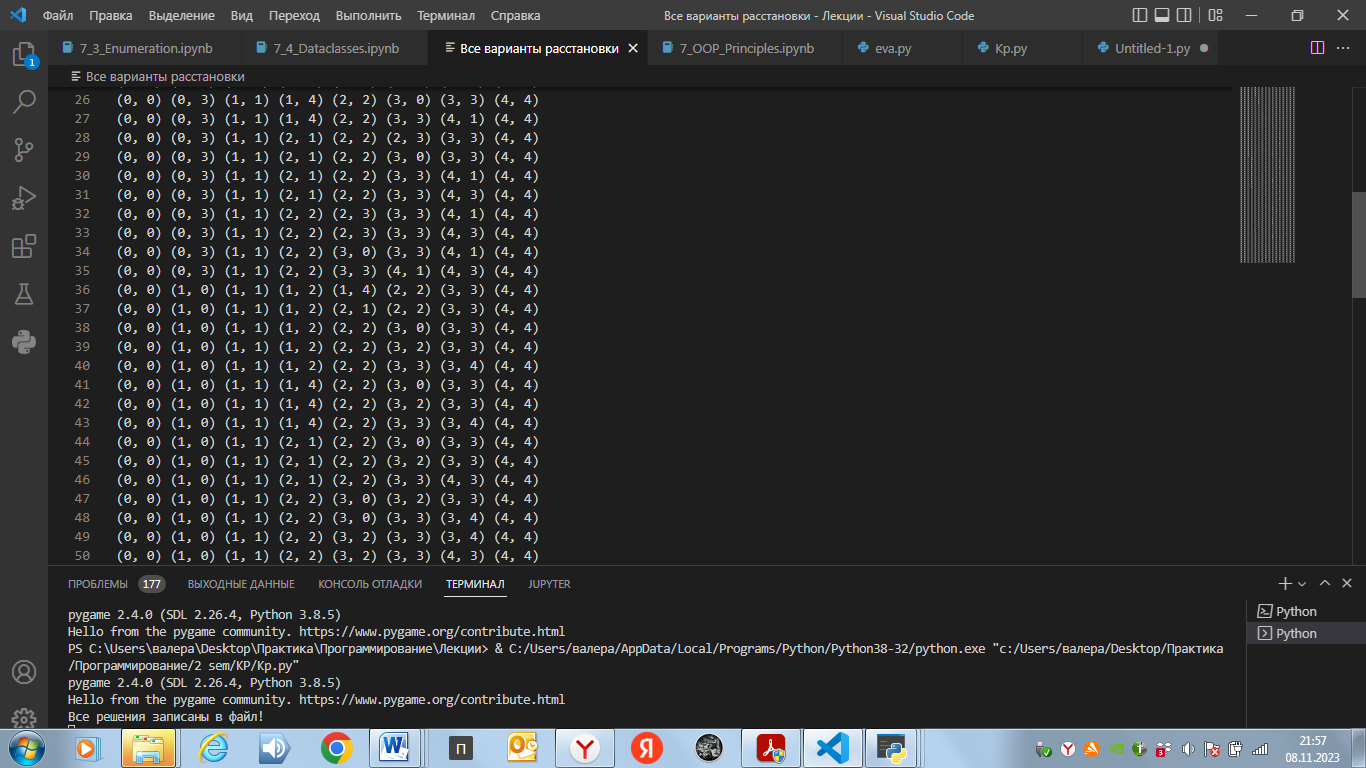
|  |  |
| --- | --- |
|  |  |

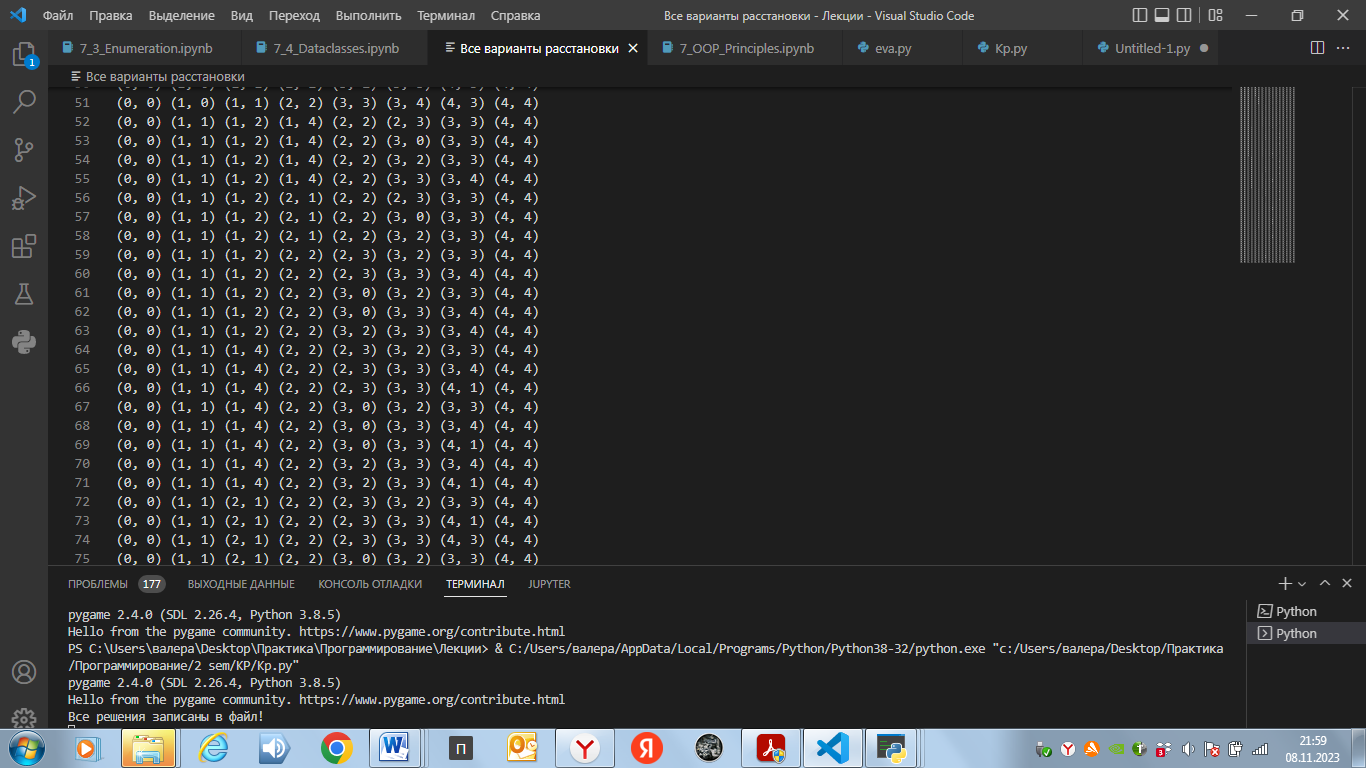
Окно с шахматной доской и кнопкой:

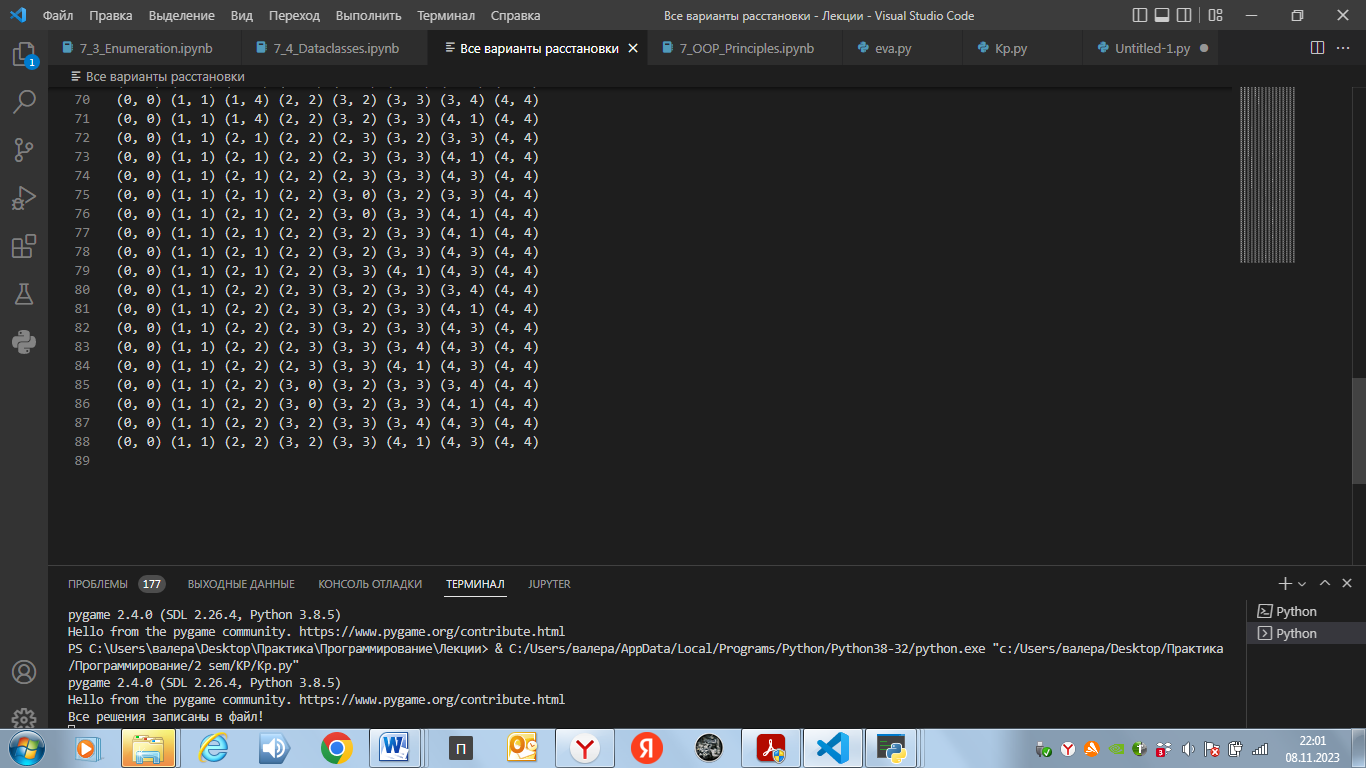


Запись всех вариантов расстановок фигур в файл:

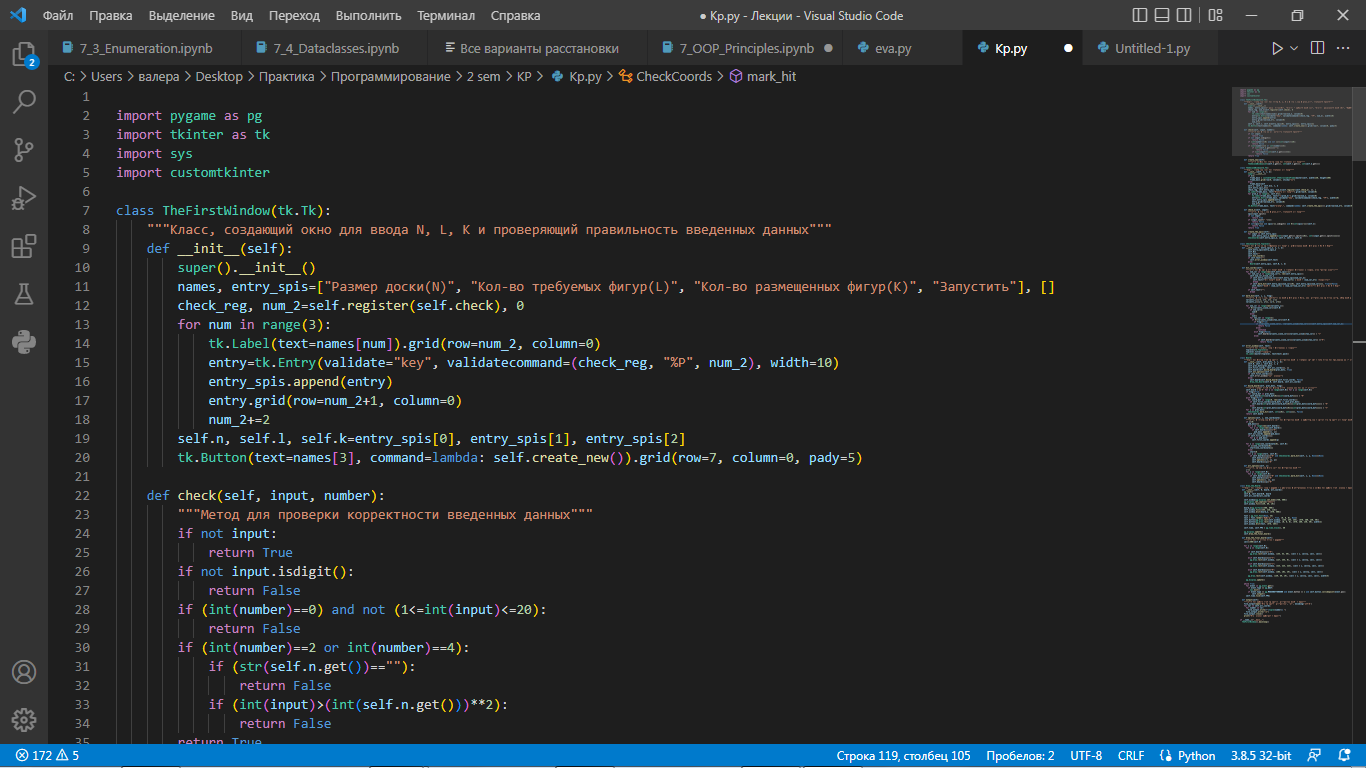


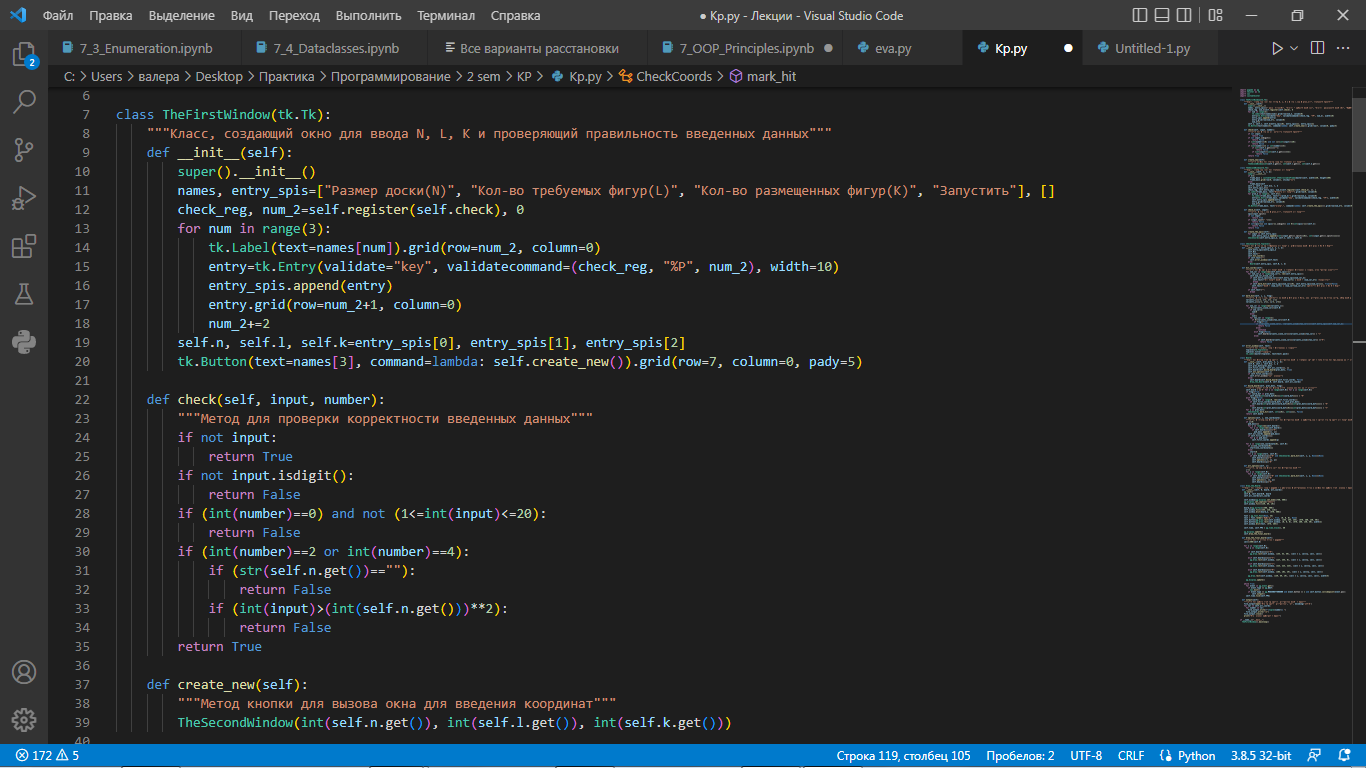


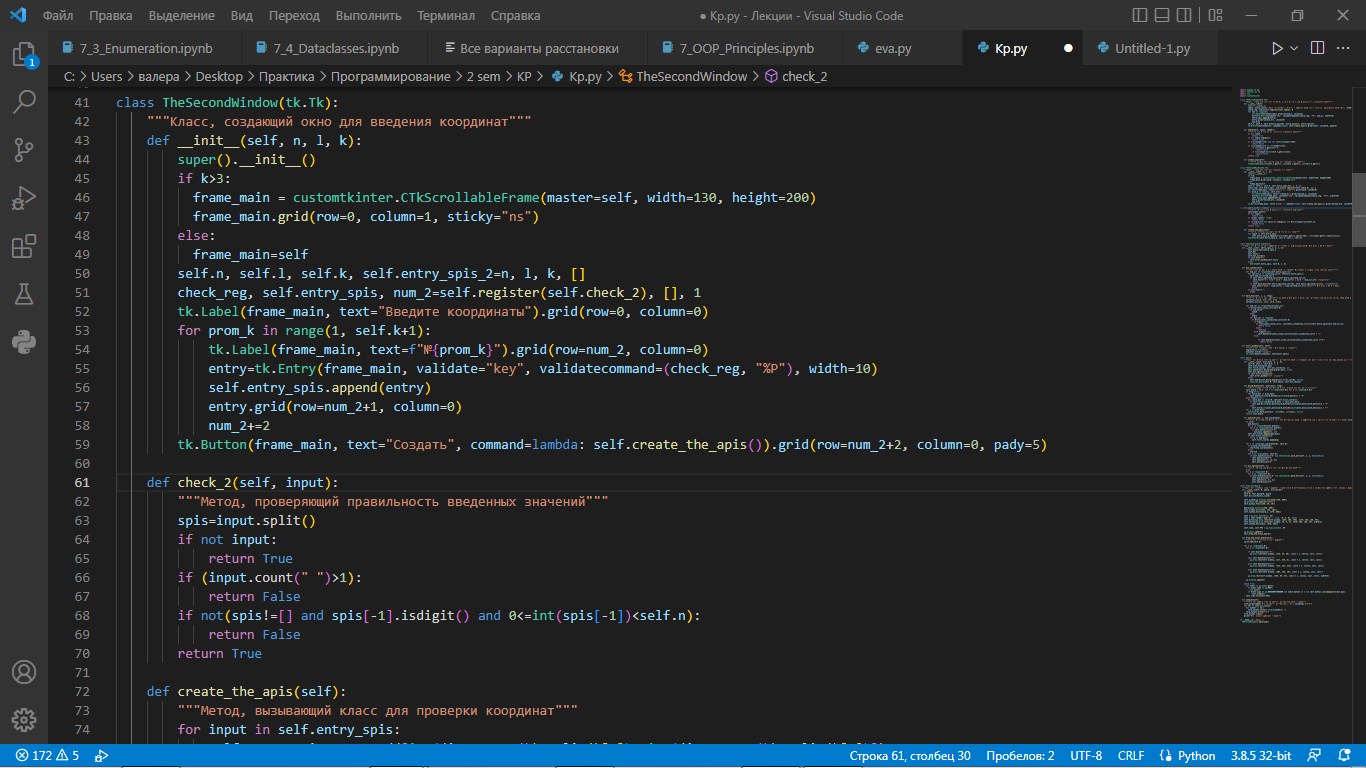


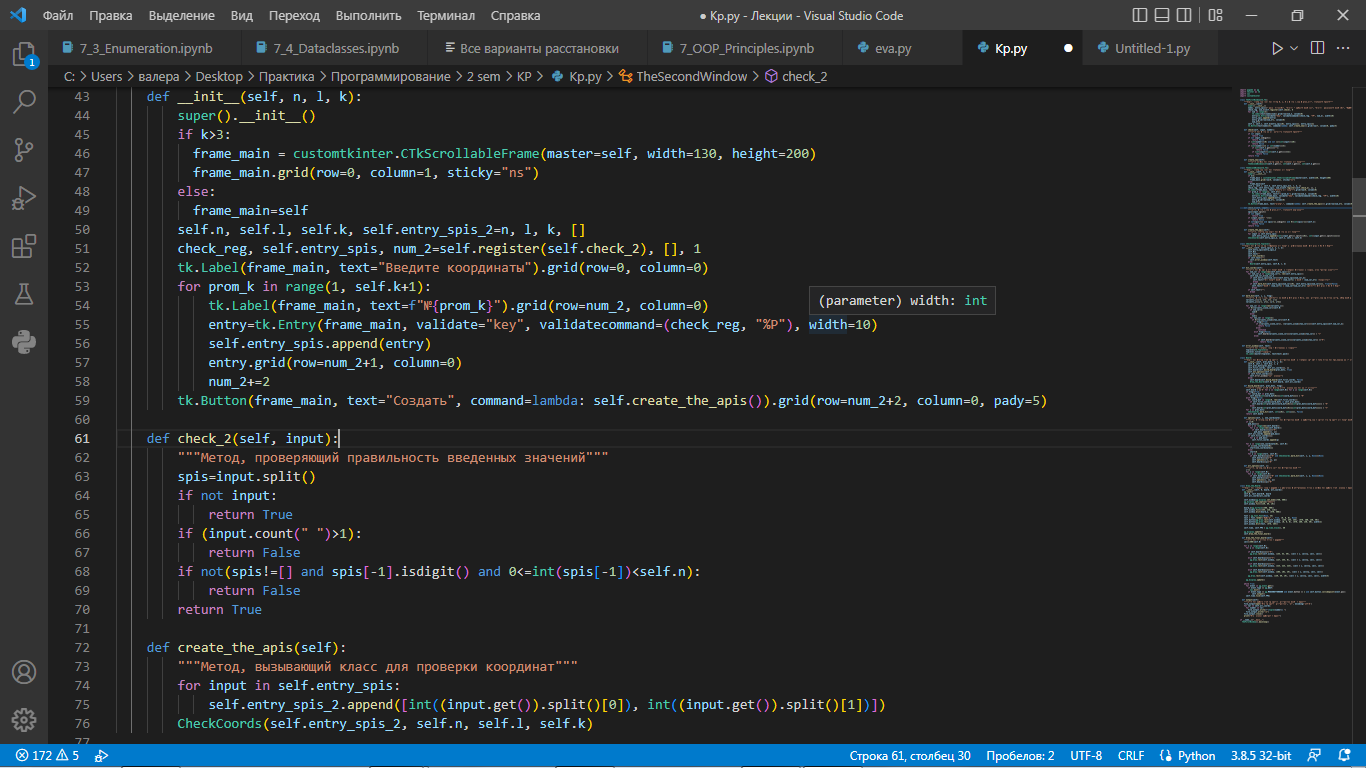


* 1. **Листинг кода.**

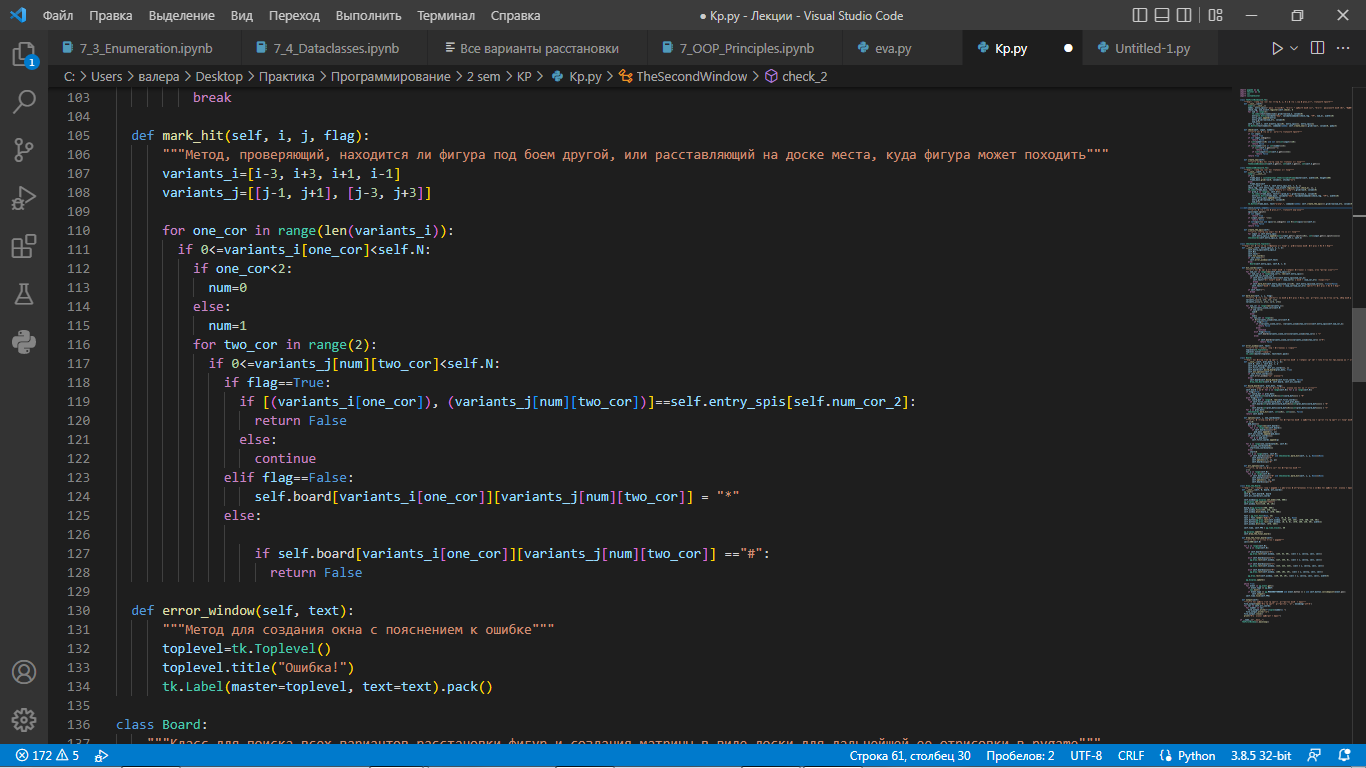


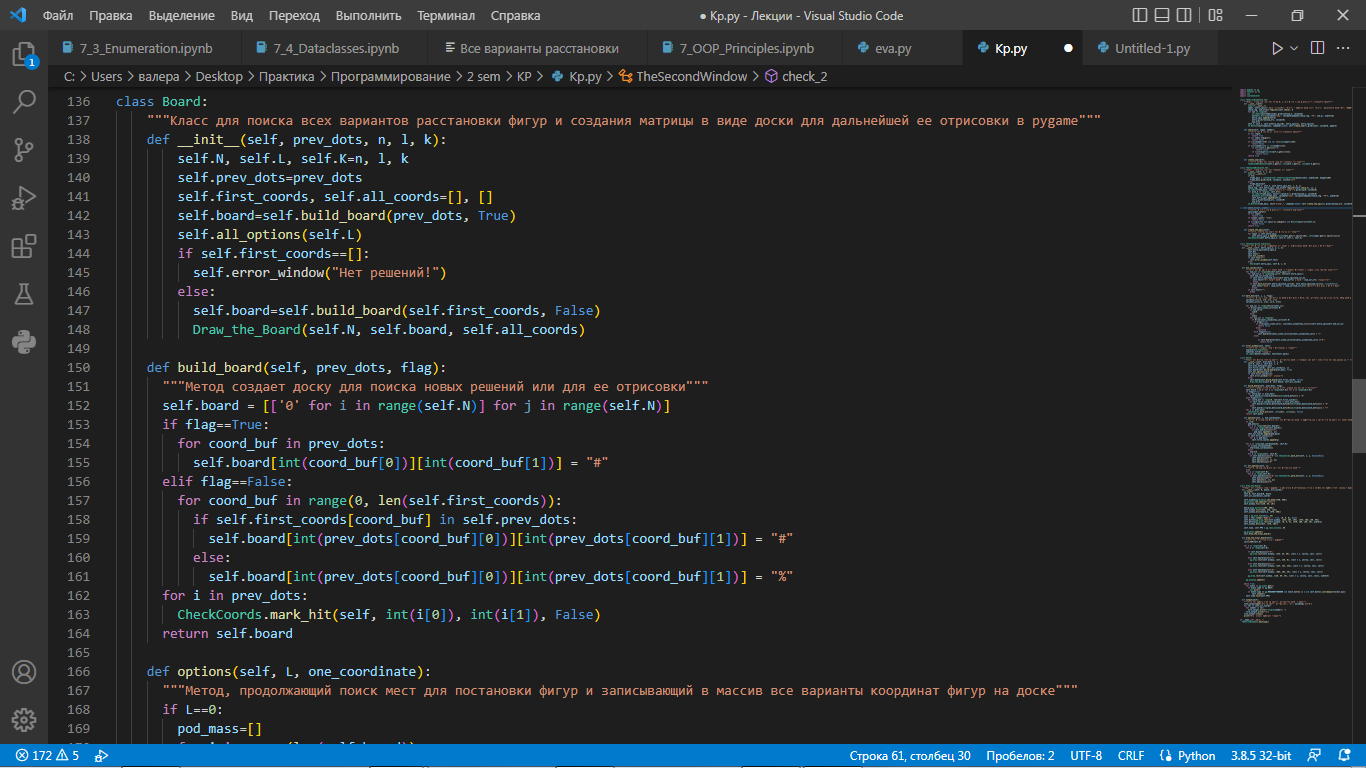


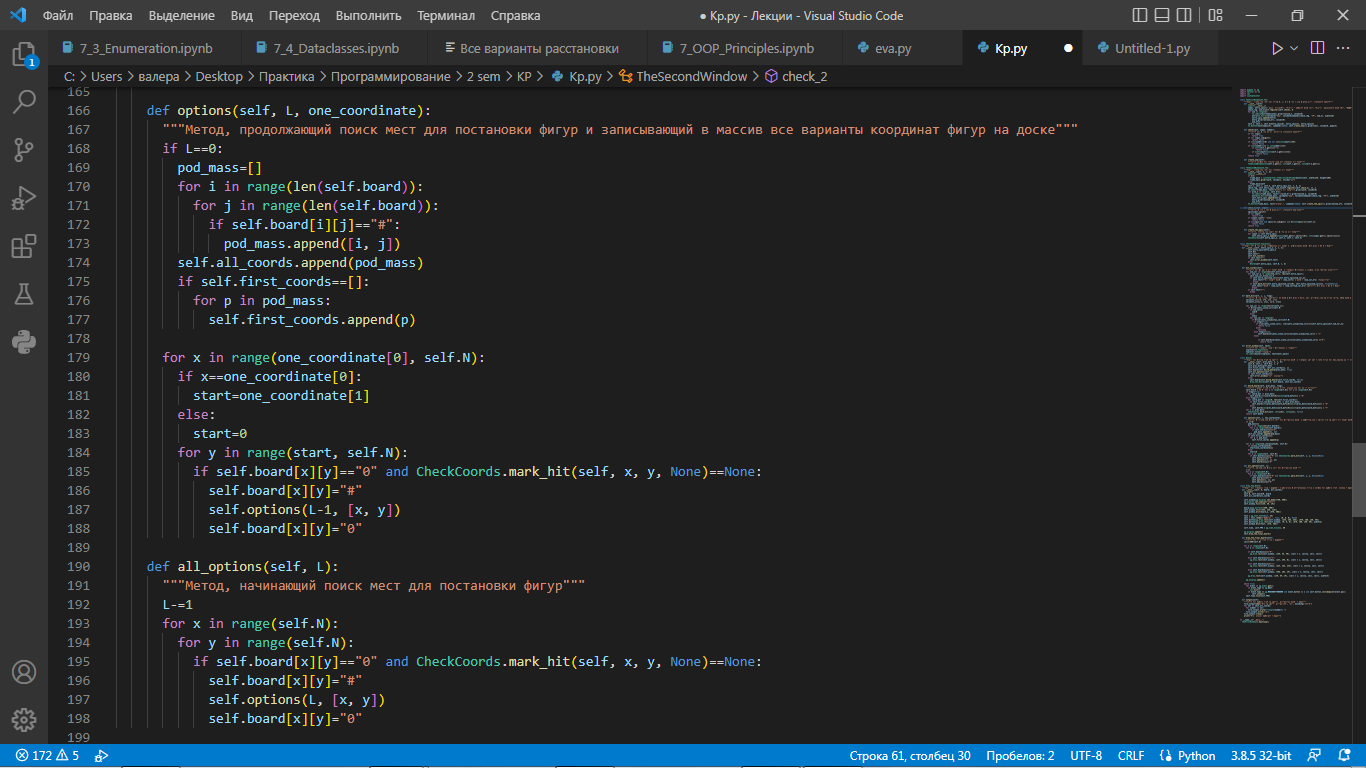


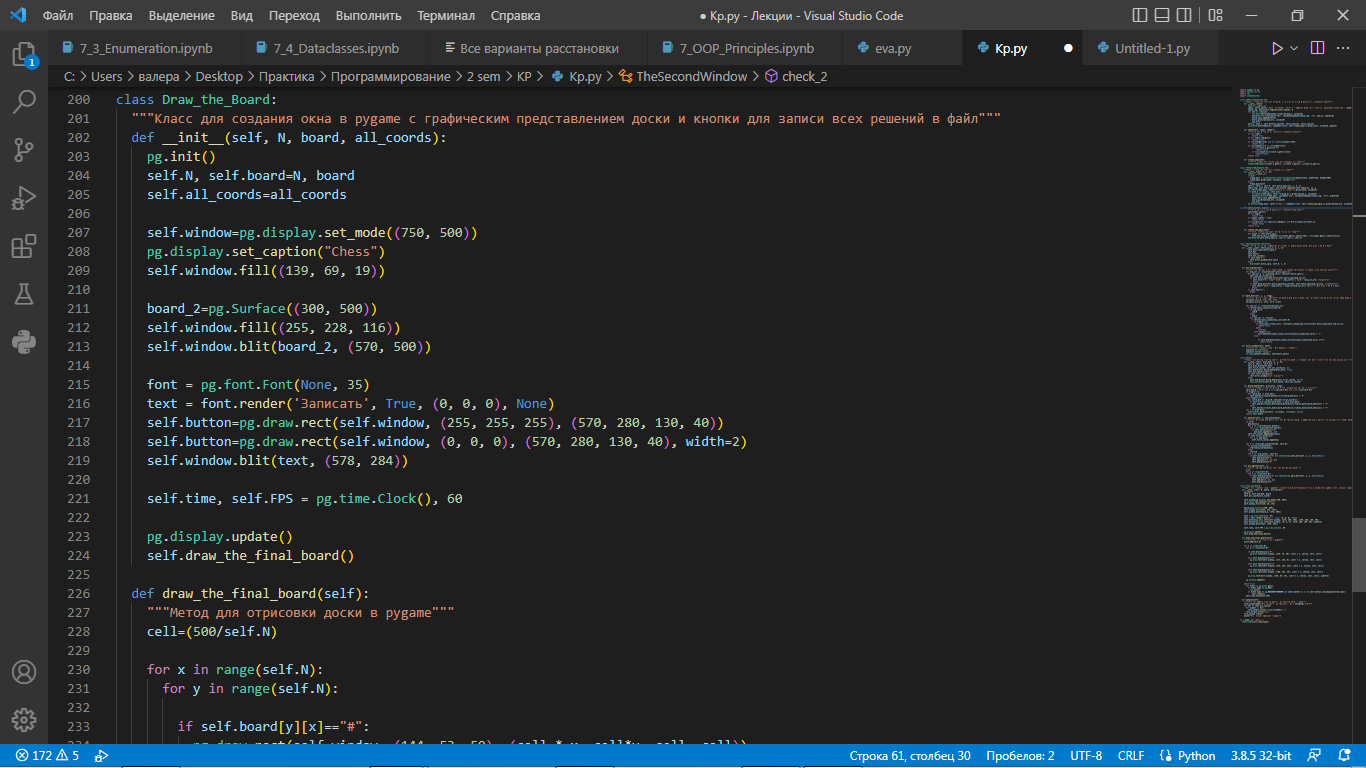


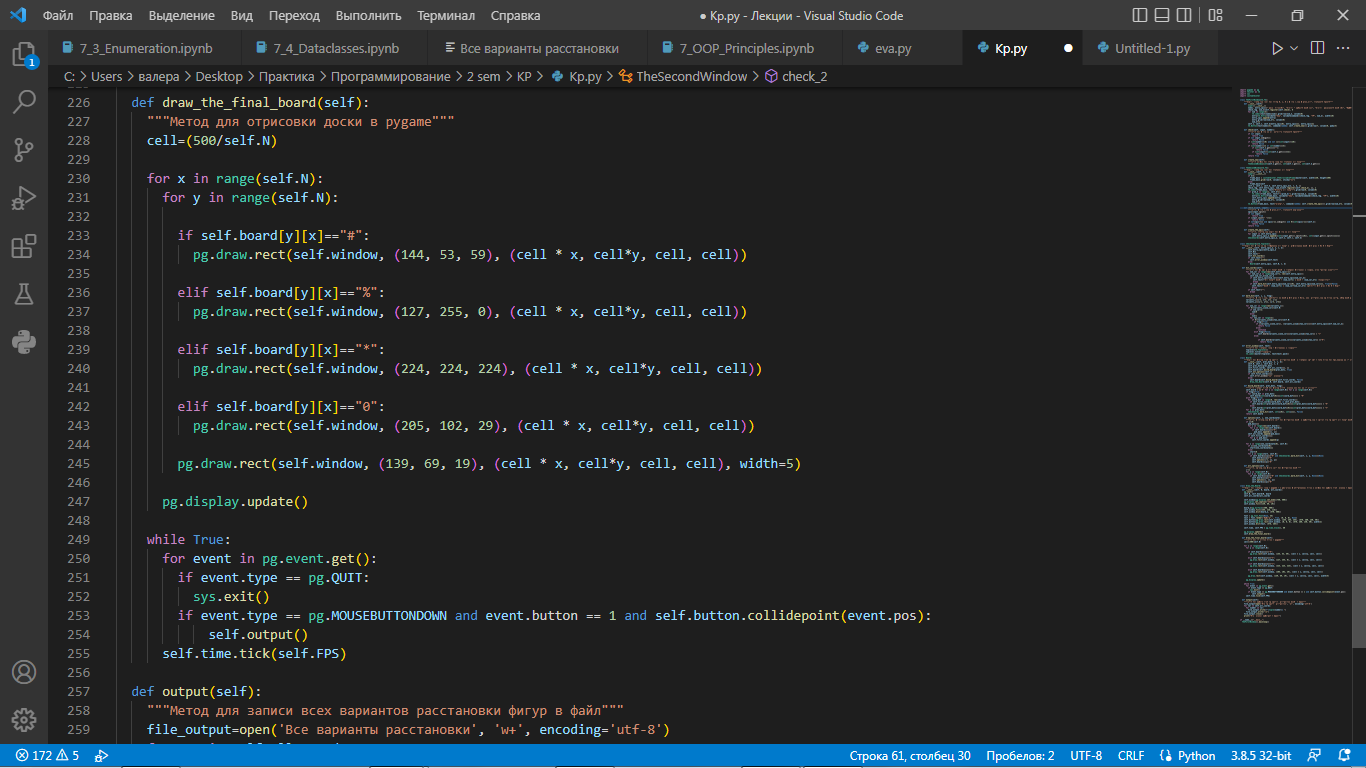


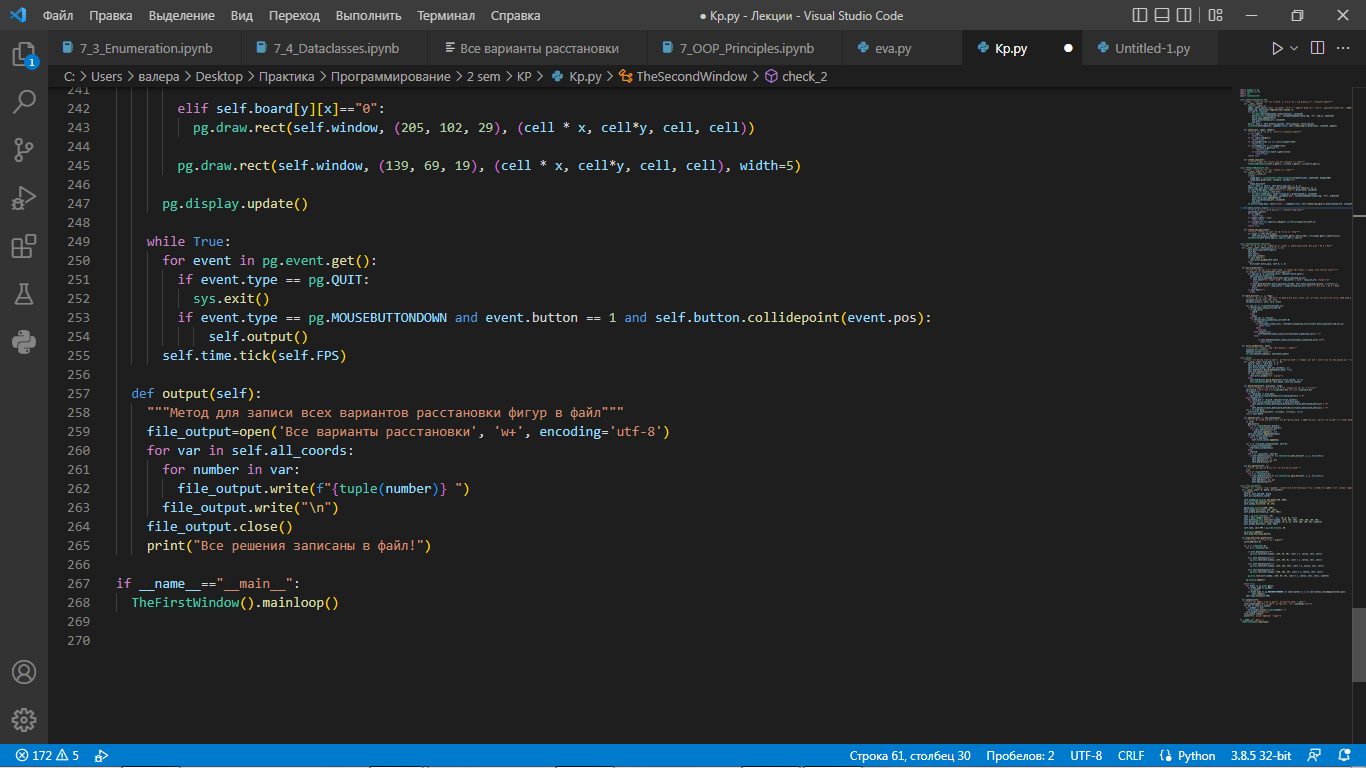












# Заключение

В результате работы над курсовым проектом я укрепила свои знания в области работы с основами функционального программирования языка Python; впервые реализовала полноценный проект с использованием основ ООП языка Python; научилась работать с UML диаграммами, создавать GUI с использованием tkinte, а также pygame.

# Список используемых источников

* <https://техномастерская.рф/customtkinter> - Customtkinter
* <https://metanit.com/python/tkinter/2.8.php> - validatecommand
* <https://habr.com/ru/articles/508710/> - UML диаграммы