Lab 5: 8-Queen Problem

Brandon Trinkle

Arizona State University

Course Number: IFT 360

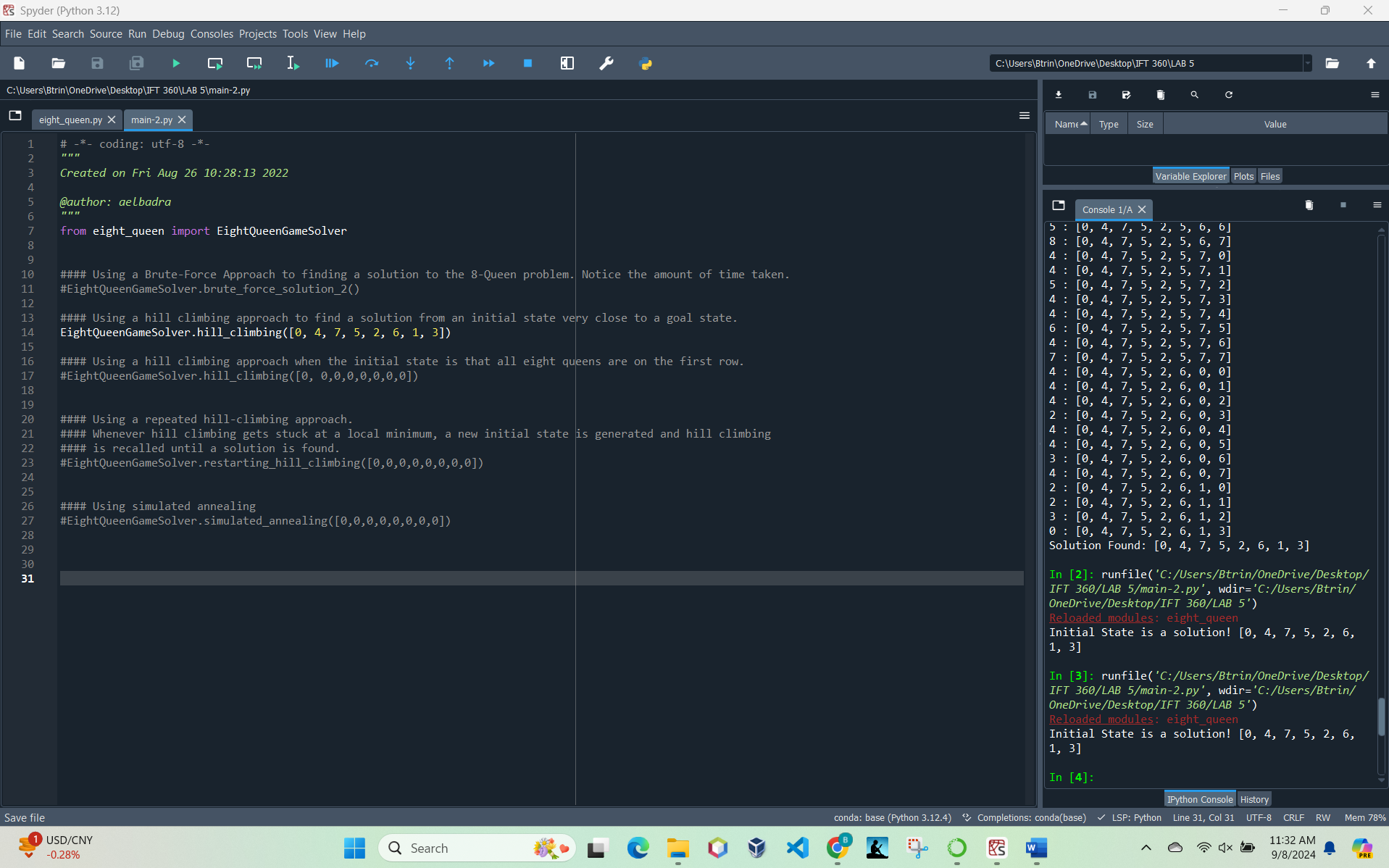
Professor Durgesh Sharma

9/8/24

**Question 1: Understanding the Code**

* **EightQueenGame Class**
  + Attributes:
    - positions: A list that holds the current positions of the queens on the board. Each index represents a row, and the value at each index represents the column position of the queen.
  + Methods:
    - \_\_init\_\_(self, p): Initializes the EightQueenGame object with a given list p, which represents the initial positions of the queens.
    - get\_num\_attacks(p): A static method that calculates the number of attacking pairs of queens for a given position list p. It iterates through each pair of queens and checks if they attack each other either horizontally, vertically, or diagonally.
    - set\_position(self, queen, p): Sets the position of a specific queen (queen index) to a new position (p) on the board.
    - set\_positions(self, p): Sets the positions of all queens to the new list p.
    - extend\_position(p): Generates all neighboring positions by moving each queen in its respective column to any other row position, producing a list of new board configurations (neighboring states).
* **EightQueenGameSolver Class**
  + Attributes:
    - There are no explicit attributes declared in the class initialization, as most methods are static or independent of object attributes.
  + Methods:
    - \_\_init\_\_(self): The constructor does not initialize any attributes but sets up the class for solving functions.
    - brute\_force\_solution(): Implements an exhaustive brute-force search by trying all possible configurations of queens using eight nested loops. It prints the current state and stops when it finds a configuration with zero conflicts.
    - brute\_force\_solution\_2(): A more concise version of the brute-force approach using the itertools.product function to generate all combinations of queen positions, checking each combination until a solution is found.
    - hill\_climbing(initial\_position): Starts with an initial position and iteratively moves to neighboring states that have fewer attacks. If no better state is found, the algorithm stops, indicating it has reached a local minimum.
    - restarting\_hill\_climbing(initial\_position): Repeatedly applies the hill climbing method, restarting with a new random initial position whenever a local minimum is reached. This process continues until a solution is found, counting the number of restarts.
    - simulated\_annealing(initial\_position): (Placeholder) Intended to solve the problem using simulated annealing, which differs from hill climbing by occasionally accepting worse states based on a probabilistic factor. The specific implementation is expected to be filled in by copying and modifying the hill climbing method according to the lab instructions.

**Question 2:** *Comment the previous method and uncomment the first hill climbing call which starts from this state: [0, 4, 7, 5, 2, 6, 1, 3]. What is the obtained result?*



This situation demonstrates a best-case scenario for the hill climbing algorithm because it started at a goal state.

**Question 3**: *Comment the first hill climbing call and uncomment the second hill climbing call that starts from this state: [0, 0, 0, 0, 0, 0, 0, 0]. What is the obtained result?*

A screenshot of a computer

Description automatically generated

EightQueenGameSolver reached a local minimum during hill climbing, unable to find the optimal solution starting from the initial state [0, 0, 0, 0, 0, 0, 0, 0]. This demonstrates a common challenge with the hill climbing approach, where it often gets stuck at local optima, as observed in the final configuration [1, 7, 2, 6, 3, 5, 0, 4].

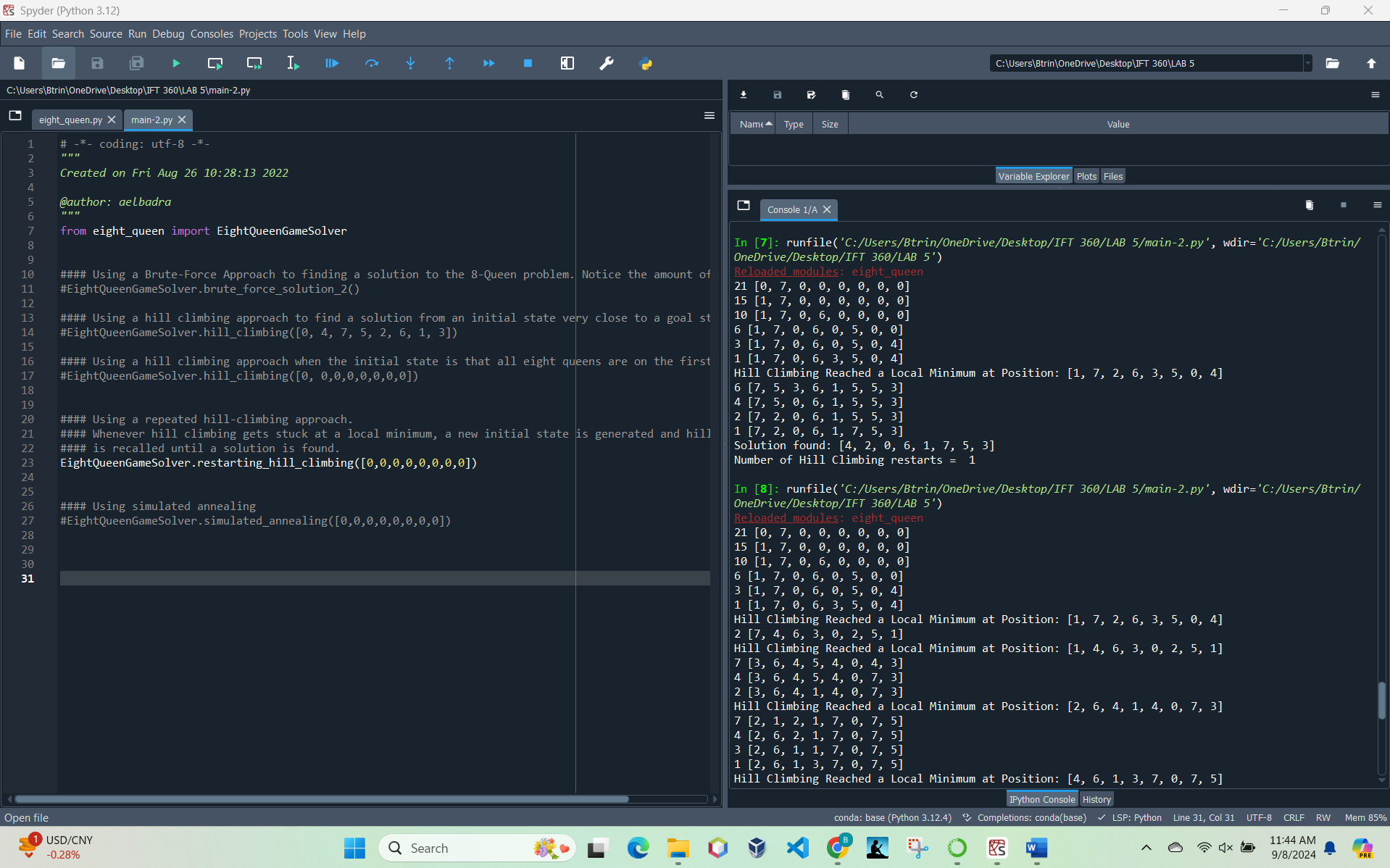
**Question 4**: *Comment the previous call and uncomment the call to the Restarting Hill Climbing method that starts from this state: [0, 0, 0, 0, 0, 0, 0, 0]. Does this method get stuck? Try running it two more times. What is the number of hill climbing restarts for each run?*

A screenshot of a computer

Description automatically generated

Yes, the Restarting Hill Climbing method does get stuck multiple times, reaching local minima. However, unlike the standard hill climbing approach, it automatically restarts with a new random initial configuration when stuck. In the run shown, the Restarting Hill Climbing method restarted 5 times before finding a solution. Restarting Hill Climbing is effective in eventually finding a solution by escaping local minima through multiple restarts. The number of restarts can vary with each execution due to the randomness of the new initial states.

**Run two more times**



Run 2 number of restarts: 1

A screenshot of a computer

Description automatically generated

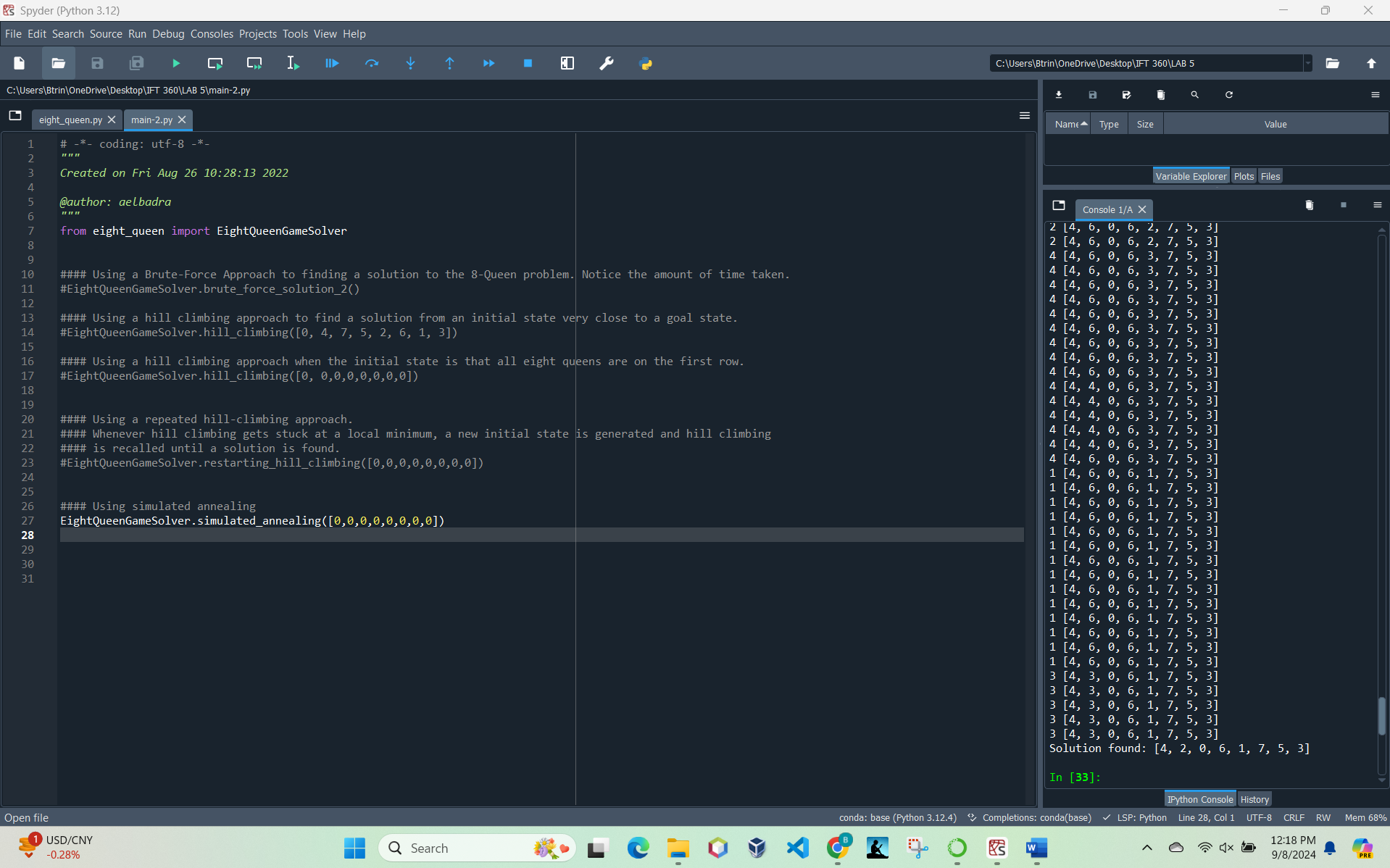
Run 3 number of restarts: 3

**Question 5:** *Code Writing for Simulated Annealing*

A screenshot of a computer

Description automatically generated

We can see that min\_attacks is no longer used, so this needs to change. It also shows an error that it is not being used in eight\_queen.py.



Code added:

A computer screen shot of a program

Description automatically generated

Since the acceptance of worse states depends on the temperature (T), the rate of cooling and the acceptance probability play a critical role. Depending on when and how often the algorithm accepts worse states, it might escape local minima faster in some runs than in others. A slow or fast acceptance of bad moves can make the time to solution vary.. These factors make each run unique in terms of the path taken and the time to reach a solution. The variability in run times and outcomes is inherent to simulated annealing due to its reliance on random decisions and probabilistic acceptance criteria. This is a strength of the algorithm because it allows exploration of the search space in ways that deterministic methods cannot, but it also means that no two runs are exactly alike.