**Foundation of Machine Learning Lab (ENSP212)**

**Lab Assignment Submitted to**

**K. R. Mangalam University**

**for**

A blue and white sign with black text

Description automatically generated

**Bachelor of COMPUTER APLICATION**

**in**

**AI & DS**

**Submitted by**

**Prabhjot Singh (2301201127)**

**Course Teacher**

**Dr. Aarti**

**School of Engineering & Technology**

**K. R. MANGALAM UNIVERSITY**

**Sohna, Haryana 122103, India**

**October 2025**

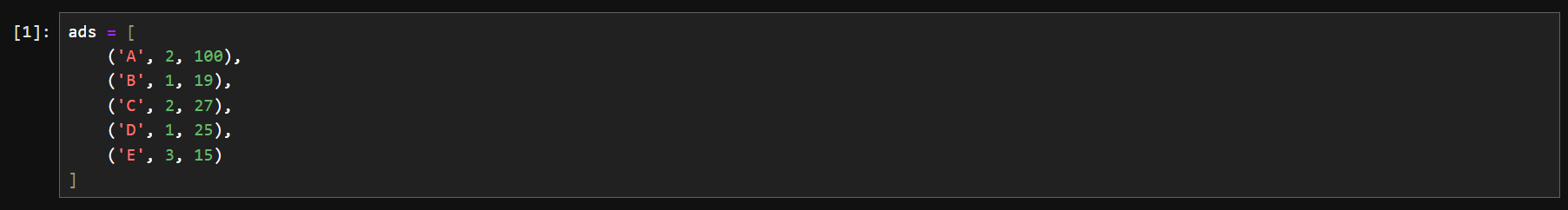
https://github.com/psbhara/ada2

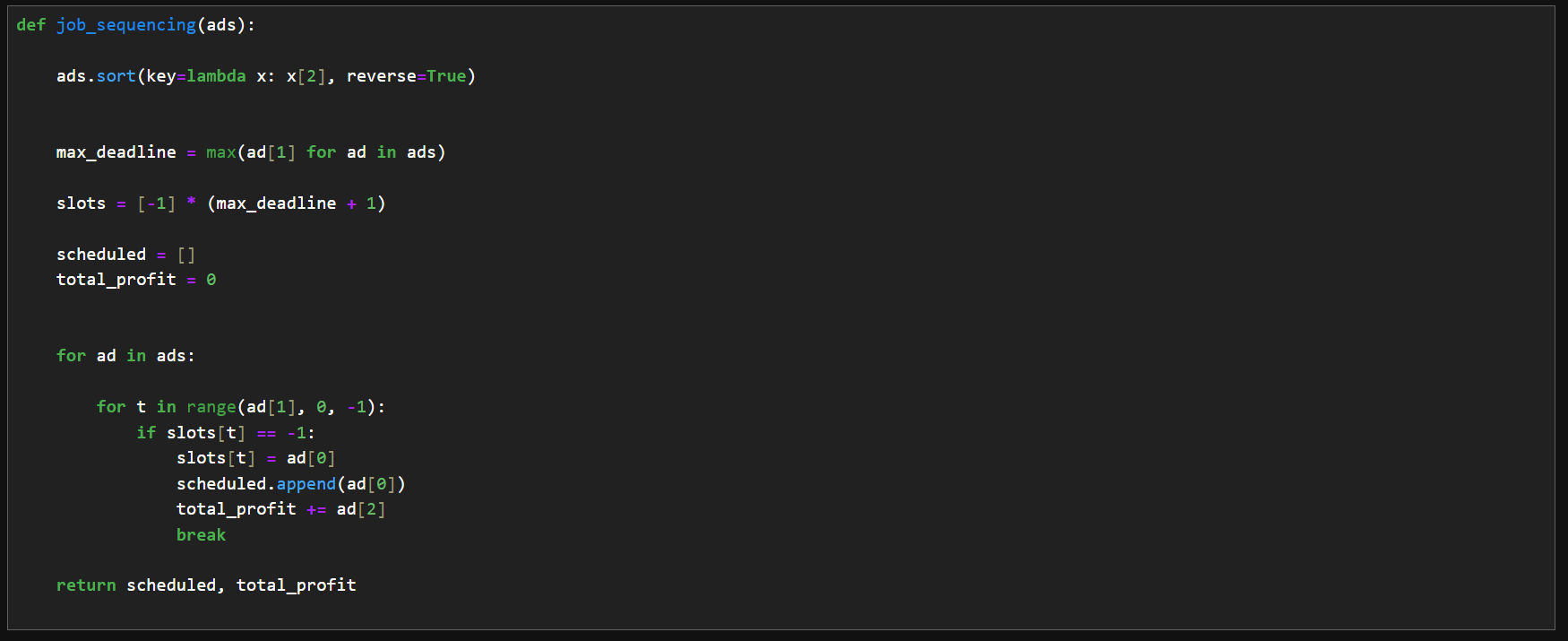
**Problem 1: Scheduling TV Commercials to Maximize Impact**

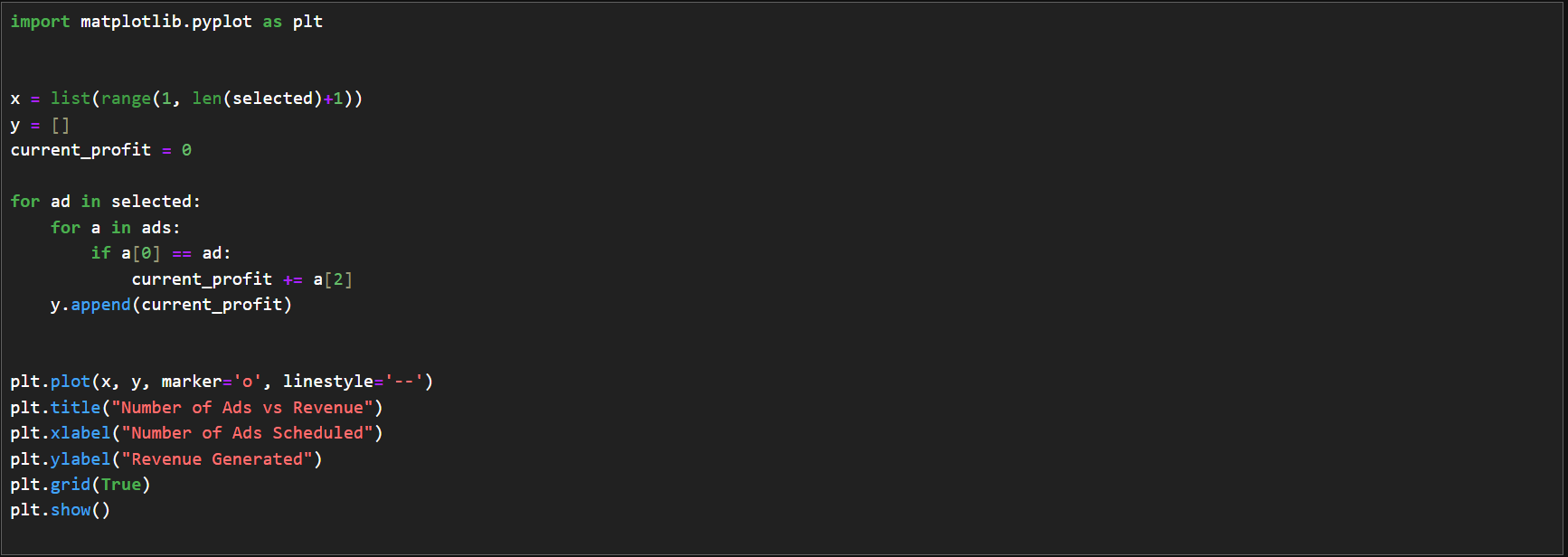
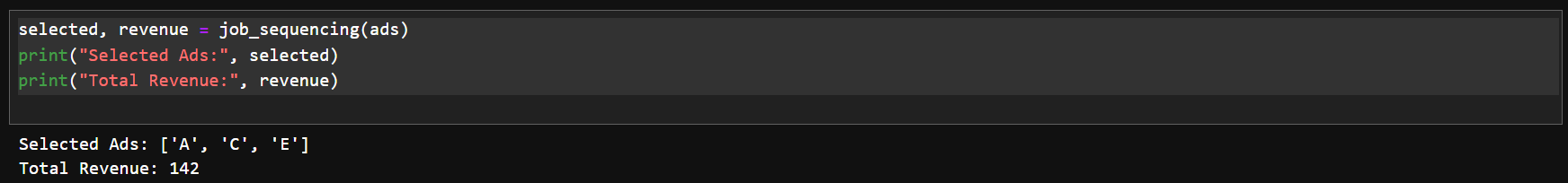
* **Algorithmic Strategy**: **Greedy (Job Sequencing)**
* **Application Domain**: Media & Advertisement
* **Problem Description**: You are given a list of commercials, each with a deadline and revenue. Schedule non-overlapping commercials to maximize total revenue.

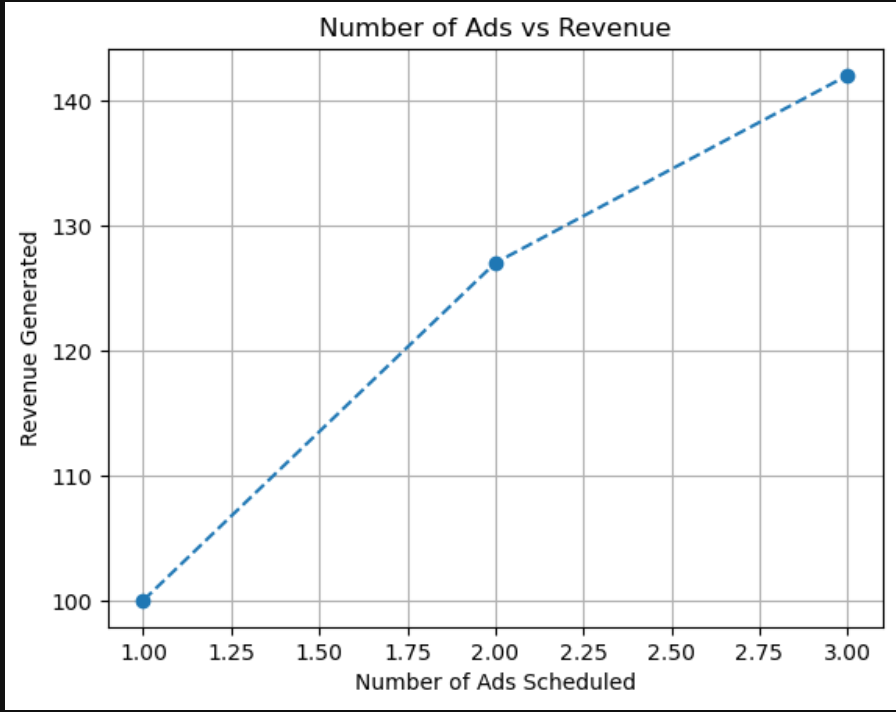
**Sub-Tasks**:

1. **Input**: A list of ads with (id, deadline, profit) values.
2. **Approach**: Use the Greedy algorithm to sort ads by profit and schedule the most profitable ones within available slots.
3. **Output**: List of selected ad slots and total revenue.
4. **Analysis**: Time and space complexity. Discuss real-world constraints like ad runtime and slot availability.
5. **Visualization**: Plot number of ads vs. revenue generated.







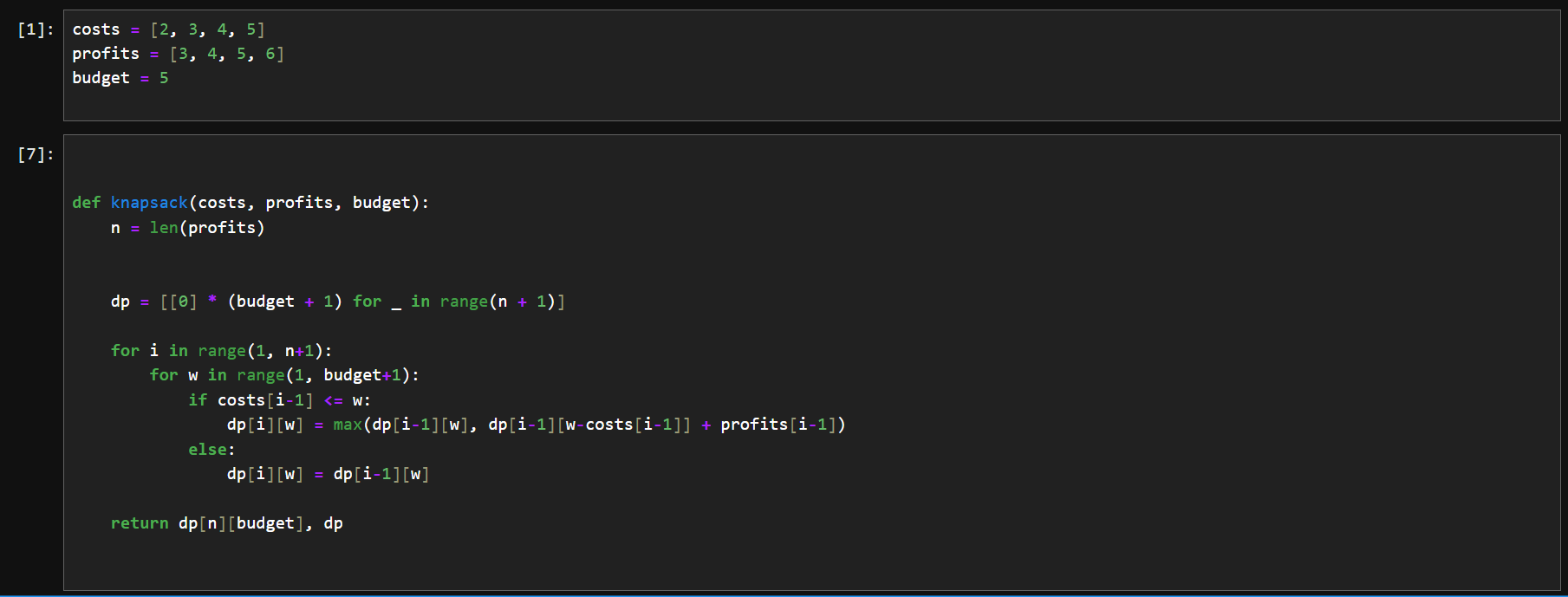


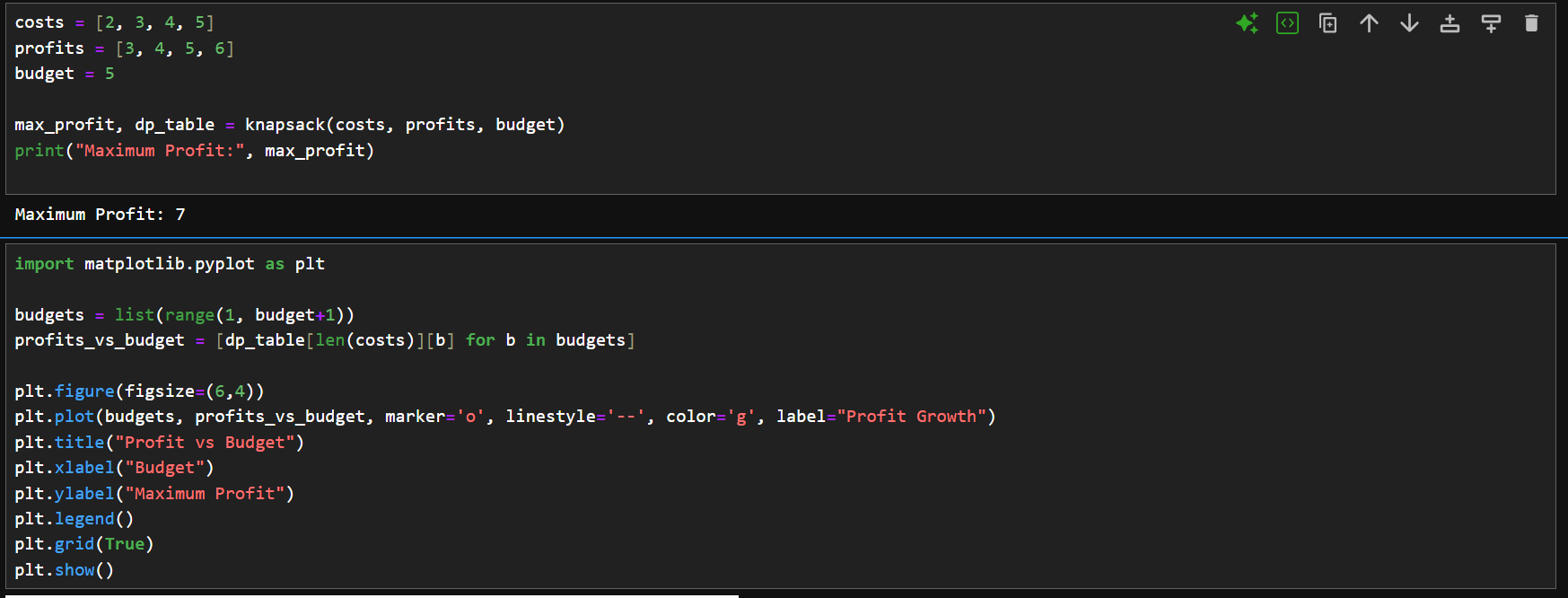
**Problem 2: Maximizing Profit with Limited Budget**

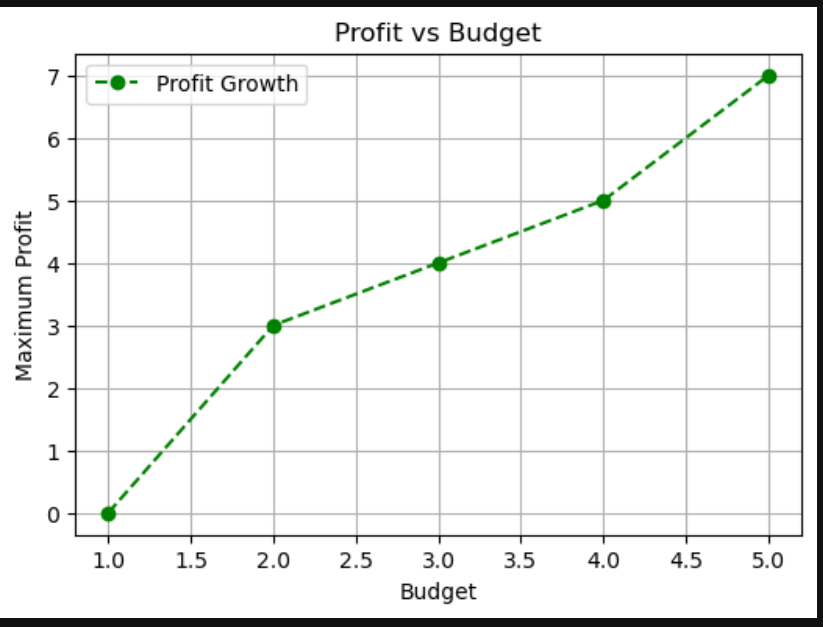
* **Algorithmic Strategy**: **Dynamic Programming (0/1 Knapsack)**
* **Application Domain**: Investment, Budget Planning
* **Problem Description**: Choose a subset of projects or items that give maximum profit within a limited budget.

**Sub-Tasks**:

1. **Input**: Lists of item weights (costs), values (profits), and total budget (capacity).
2. **Approach**: Use a bottom-up 0/1 Knapsack dynamic programming algorithm.
3. **Output**: Maximum profit achievable within the budget.
4. **Analysis**: Time and space complexity.
5. **Visualization**: Plot profit vs. budget or number of items.





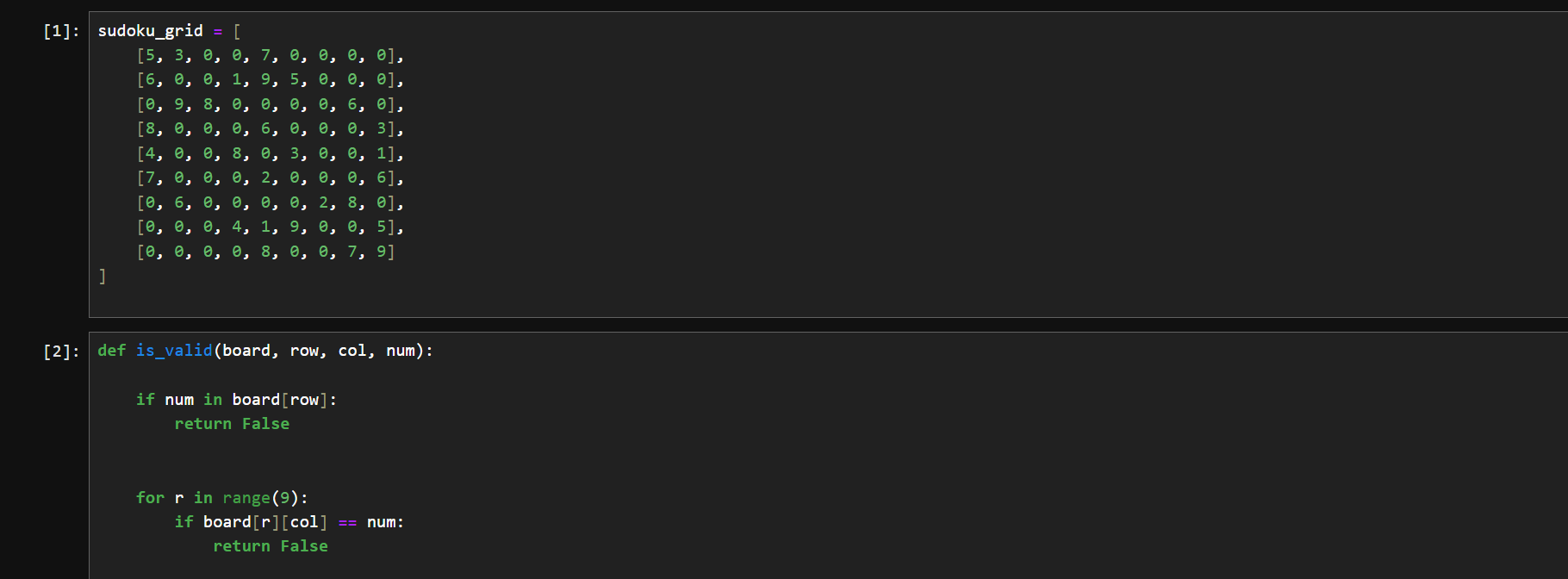


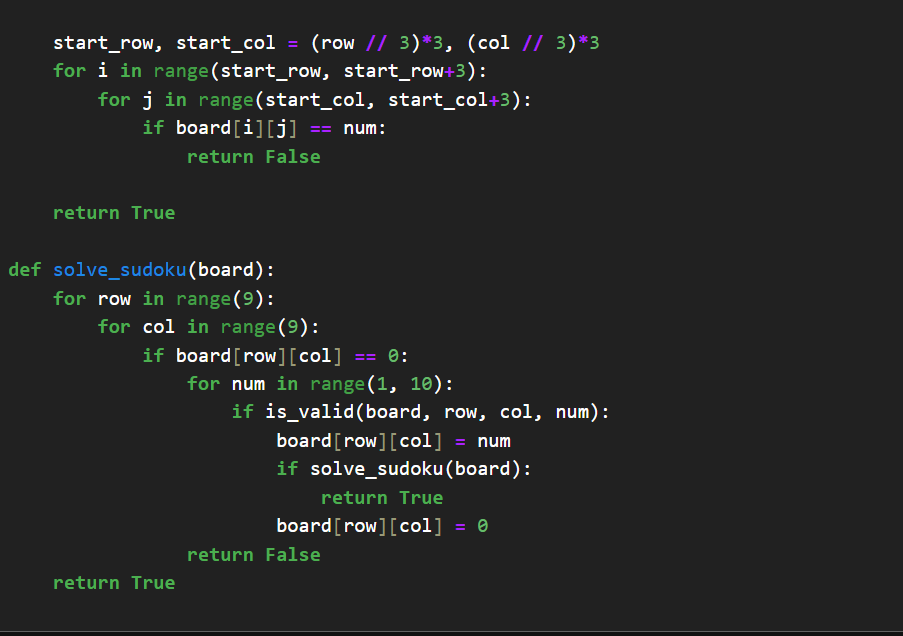
**Problem 3: Solving Sudoku Puzzle**

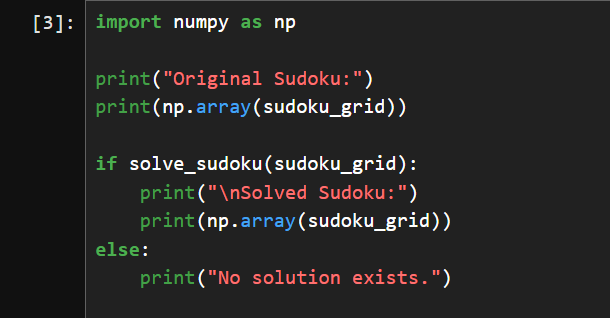
* **Algorithmic Strategy**: **Backtracking**
* **Application Domain**: Gaming, Puzzle Solvers
* **Problem Description**: Fill a 9x9 Sudoku grid such that each row, column, and 3x3 box contains all digits from 1 to 9.

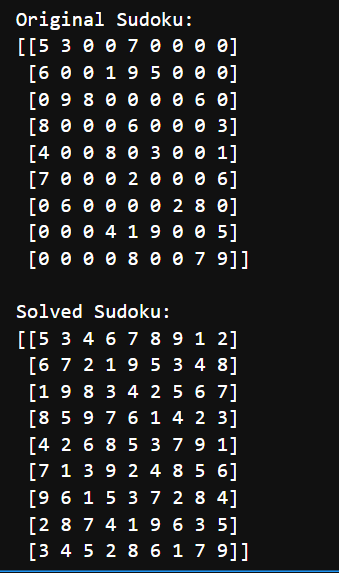
**Sub-Tasks**:

1. **Input**: A partially filled 9x9 Sudoku grid (use a sample input).
2. **Approach**: Implement recursive backtracking with constraint checks.
3. **Output**: Completed Sudoku grid.
4. **Analysis**: Discuss performance impact for complex puzzles.
5. **Visualization**: Optional – time vs. number of empty cells.







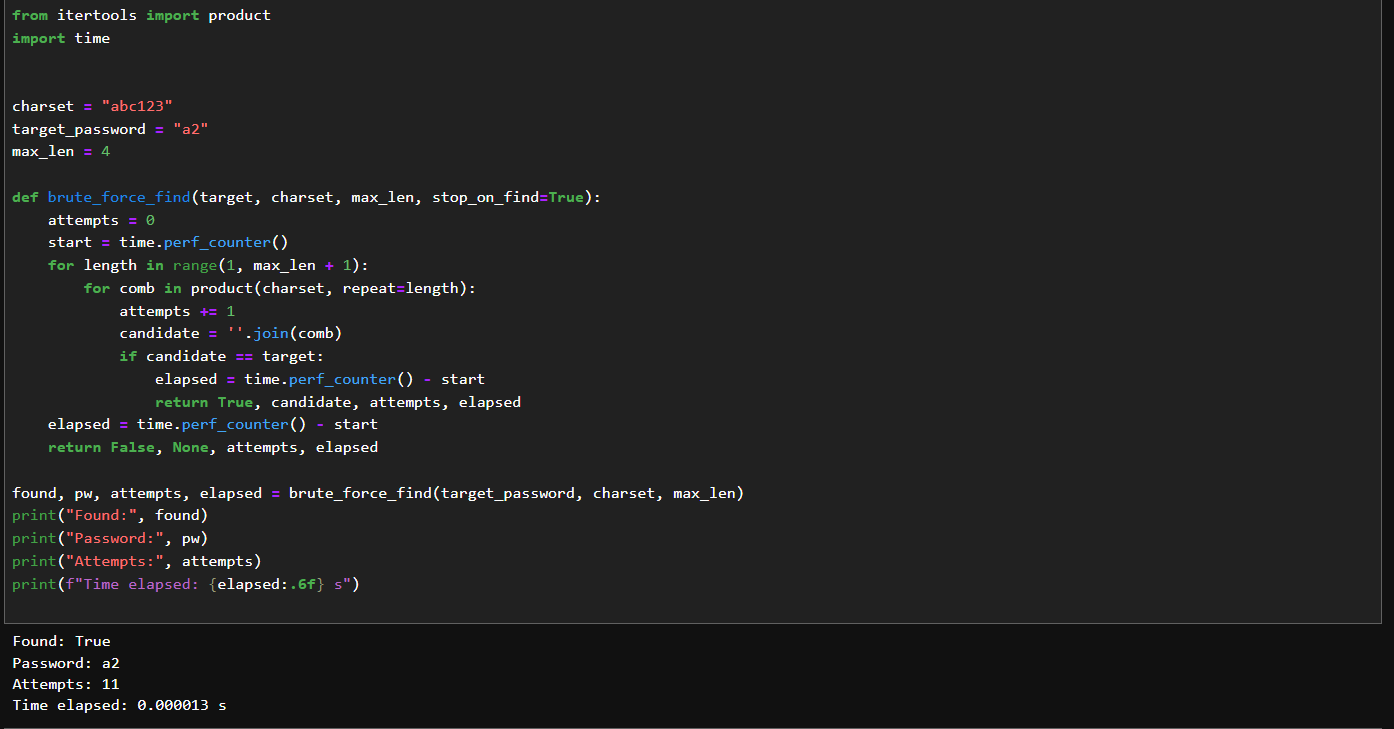


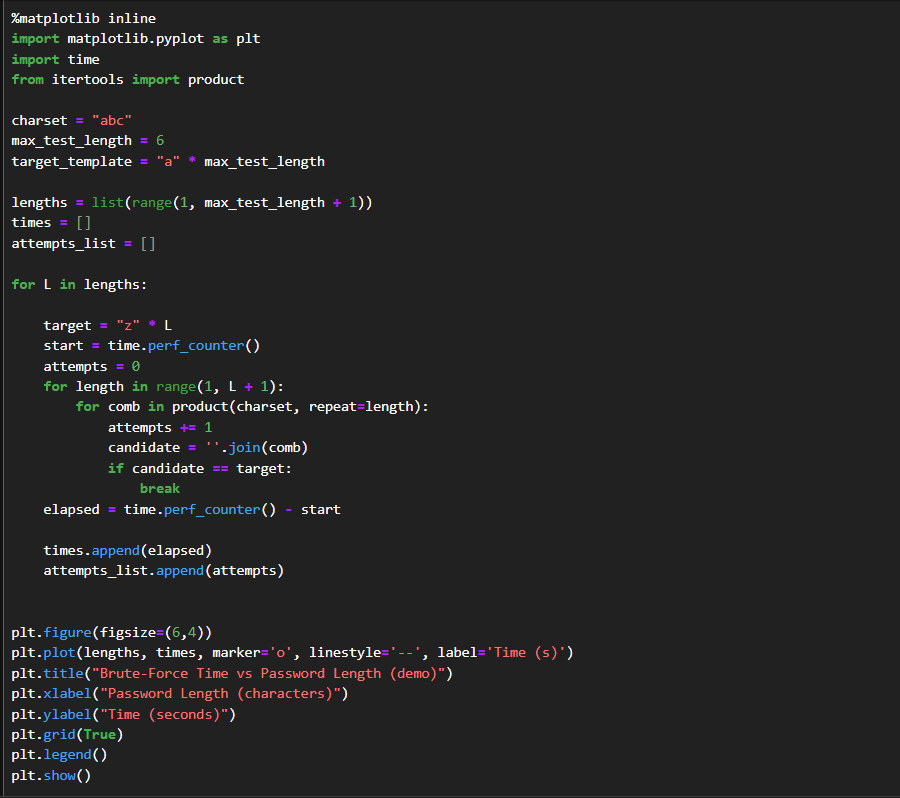
**Problem 4: Password Cracking (Naive)**

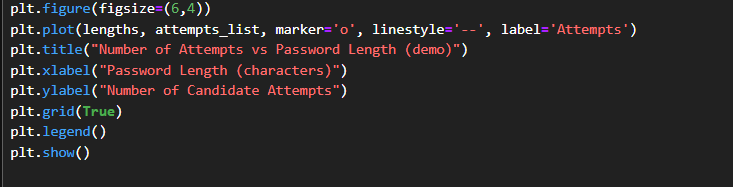
* **Algorithmic Strategy**: **Brute-Force**
* **Application Domain**: Cybersecurity
* **Problem Description**: Attempt to crack a given password using all possible character combinations from a known charset.

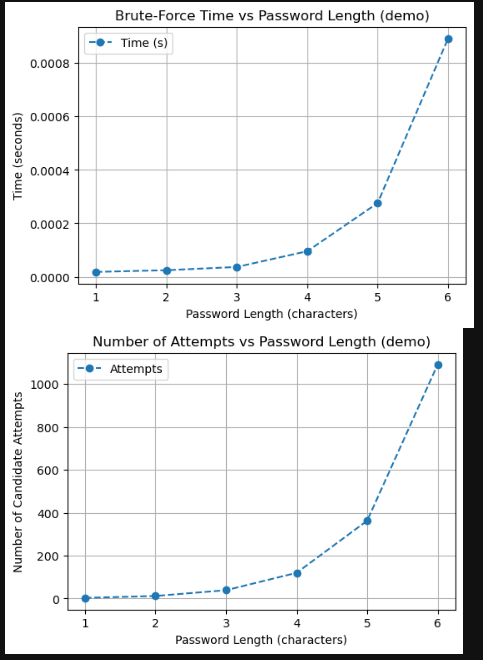
**Sub-Tasks**:

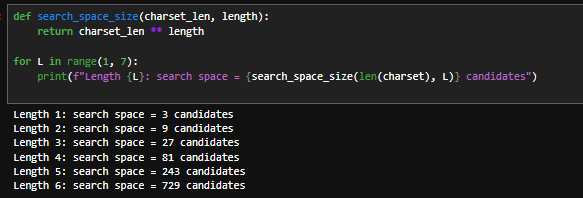
1. **Input**: Target password and character set (e.g., abc123).
2. **Approach**: Use itertools.product to try every possible combination.
3. **Output**: Matched password and number of attempts.
4. **Analysis**: Time complexity vs. password length and charset size.
5. **Visualization**: Plot time taken vs. password length.











Task 3: Experimental Profiling & Visualization For each problem:

• Use time and memory\_profiler to profile performance.

• Plot time taken and memory used with increasing input sizes (e.g., number of ads, budget items, Sudoku blanks, password length).

• Interpret graphs and describe the impact of algorithmic strategy.

• Comment on stack usage for recursive/backtracking problems.

ANS:-

**Profiling Summary**

**Tools:** time + memory\_profiler

* **Job Sequencing (Greedy)**
  + Time grows ~ O(n log n) due to sorting.
  + Memory grows ~ O(n) (slots + ads).
* **Knapsack (DP)**
  + Time & memory scale with O(n \* budget).
  + Larger budgets/items → bigger DP table → high memory.
* **Sudoku (Backtracking)**
  + Time grows **exponentially** with blanks.
  + Stack/recursion depth ≈ number of empty cells.
  + Risk of hitting recursion limit for hard puzzles.
* **Password Brute-force**
  + Time grows **exponentially** with length (|charset|^L).
  + Memory small (candidates generated on the fly).

**Visualization (what plots show)**

* **Greedy:** smooth upward curve (n log n).
* **Knapsack:** nearly linear vs. budget.
* **Sudoku:** steep jumps with more blanks.
* **Brute-force:** exponential explosion in time vs. length.