PYTHON TASK - 5

Python Developer Internship Report

Main Flow Services and Technologies Pvt. Ltd.

Name: Khushi Maurya

Task 33: Find All Permutations of a String

Objective:
Generate all permutations of a given string.

Input:
A string (e.g., "abc")

Output:
All permutations (e.g., ['abc', 'acb', 'bac', 'bca', 'cab', 'cba'])

Explanation:
We use the itertools.permutations function or recursion to generate all possible rearrangements of the characters in the string.

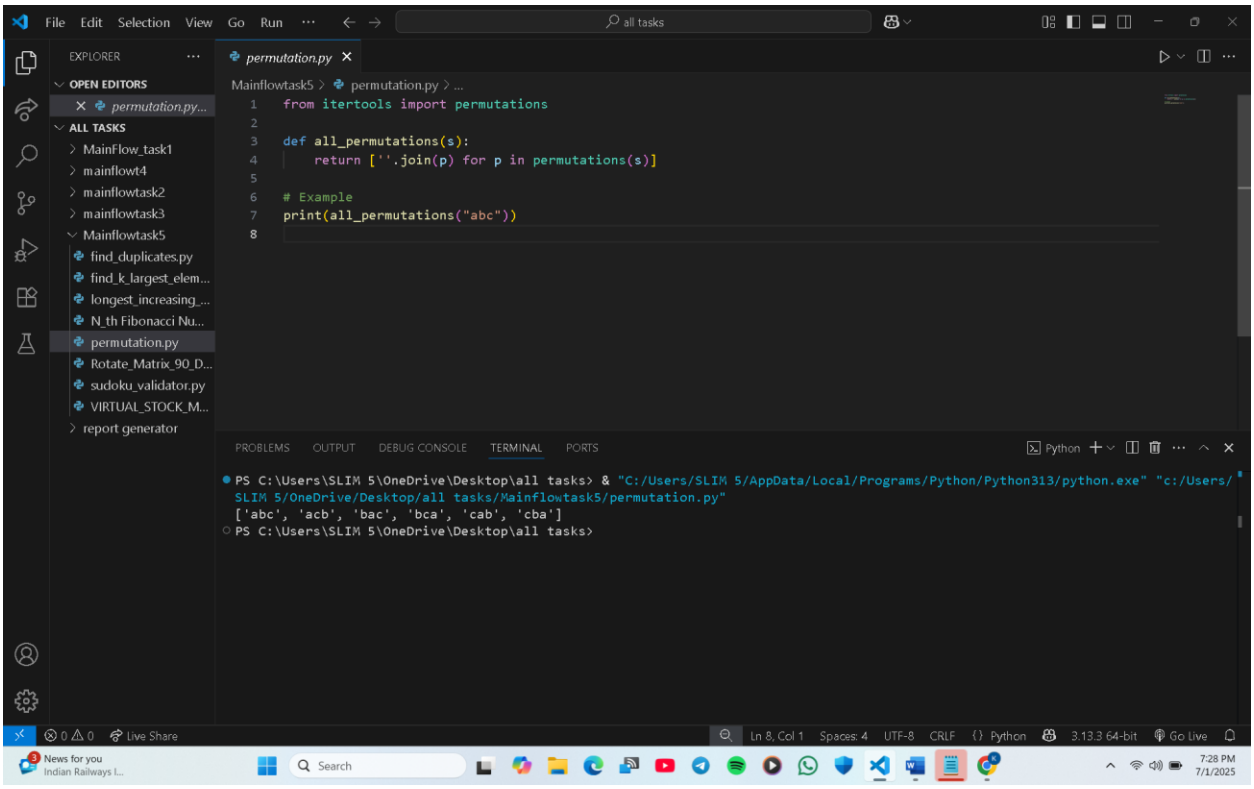
Python Code:

```
from itertools import permutations


def get_permutations(s):
    return [''.join(p) for p in permutations(s)]


print(get_permutations("abc"))
```

Screenshot:



Task 34: N-th Fibonacci Number (Dynamic Programming)

Objective:

Find the N-th Fibonacci number efficiently using dynamic programming.

Input:

An integer n

Output:

The n-th Fibonacci number

Explanation:

Using a bottom-up approach with memoization avoids repeated calculations.

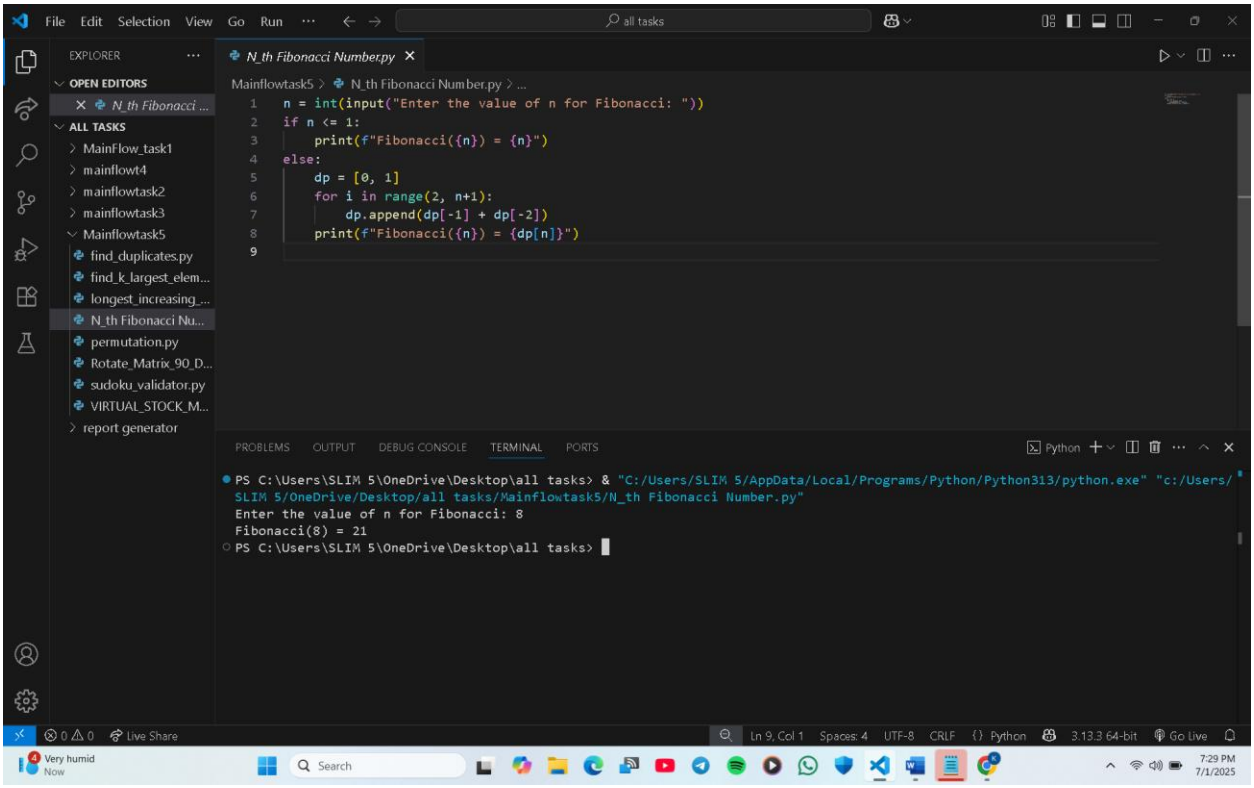
Python Code:

```
def fibonacci(n):
    if n <= 1:
        return n
    fib = [0] * (n+1)
```

```
    fib[1] = 1
    for i in range(2, n+1):
        fib[i] = fib[i-1] + fib[i-2]
    return fib[n]

print(fibonacci(10))
```

Screenshot:



Task 35: Find Duplicates in a List

Objective:
Identify duplicate elements in a list.

Input:
A list of integers

Output:
List of duplicates

Explanation:

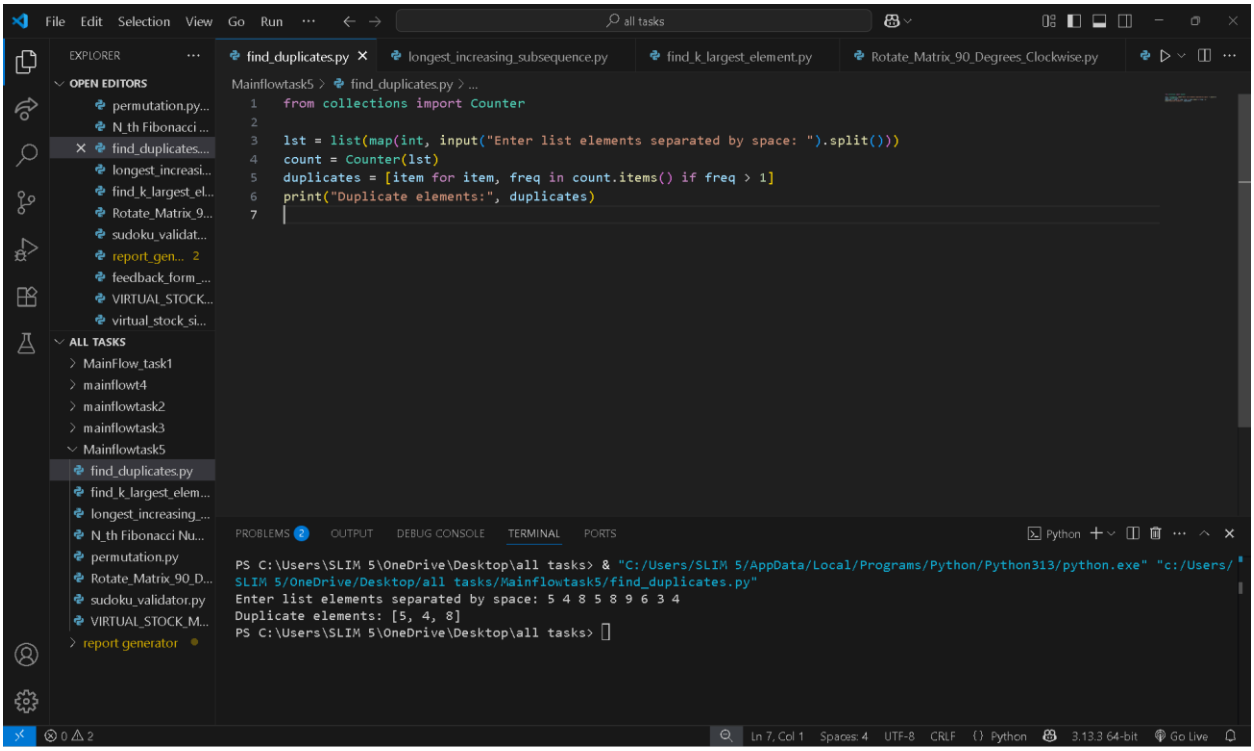
We use collections.Counter to count each element’s occurrences.

Python Code:

```
from collections import Counter


def find_duplicates(lst):
    count = Counter(lst)
    return [item for item, c in count.items() if c > 1]


print(find_duplicates([1, 2, 3, 2, 4, 5, 1]))
```



Task 36: Longest Increasing Subsequence (LIS)

Objective:

Find the length of the longest increasing subsequence in a list.

Input:
List of integers

Output:
Length of LIS

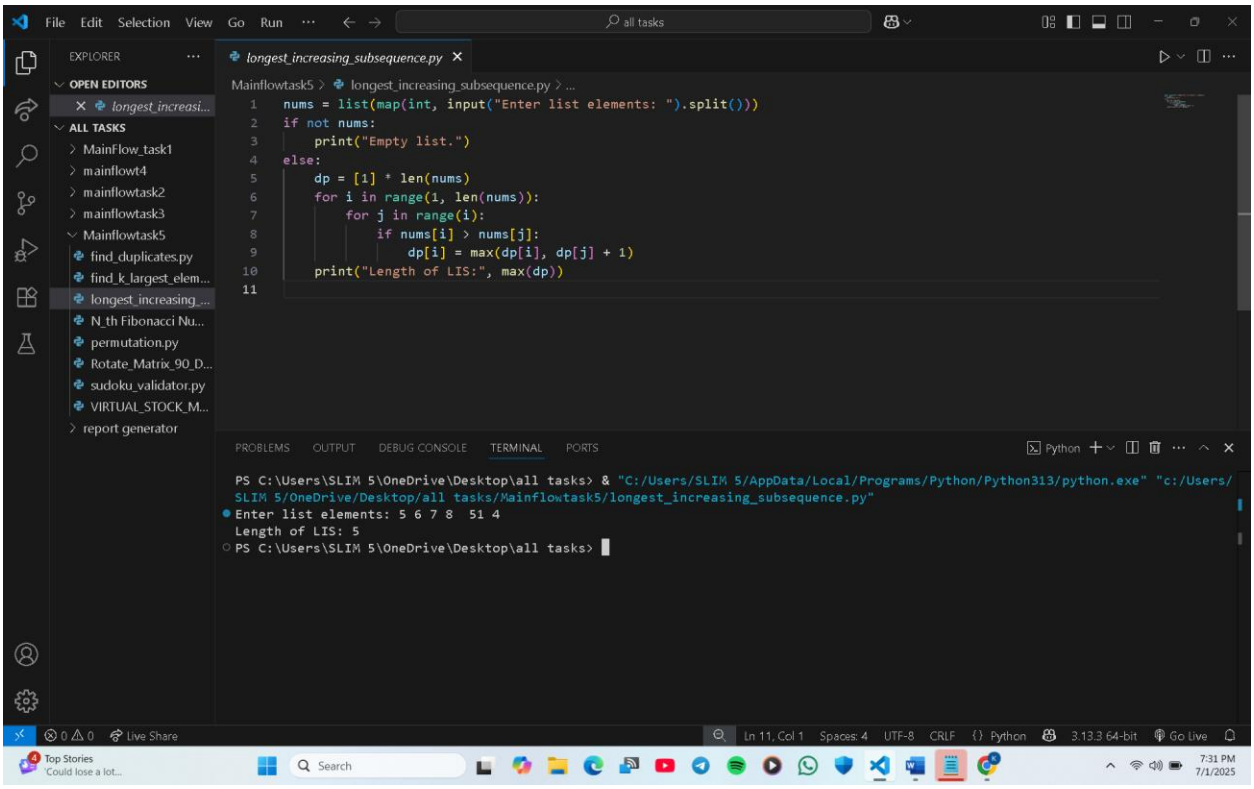
Explanation:
Dynamic programming is used to track increasing sequences.

Python Code:

```
def LIS(arr):
    n = len(arr)
    lis = [1]*n
    for i in range(1, n):
        for j in range(0, i):
            if arr[i] > arr[j] and lis[i] < lis[j]+1:
                lis[i] = lis[j]+1
    return max(lis)


print(LIS([10, 22, 9, 33, 21, 50, 41, 60]))
```

Screenshot:



Task 37: Find K Largest Elements

Objective:
Find the k largest elements in a list.

Input:
List of integers and an integer k

Output:
List of k largest elements

Explanation:
We use the heapq.nlargest() function or sort the list.

Python Code:

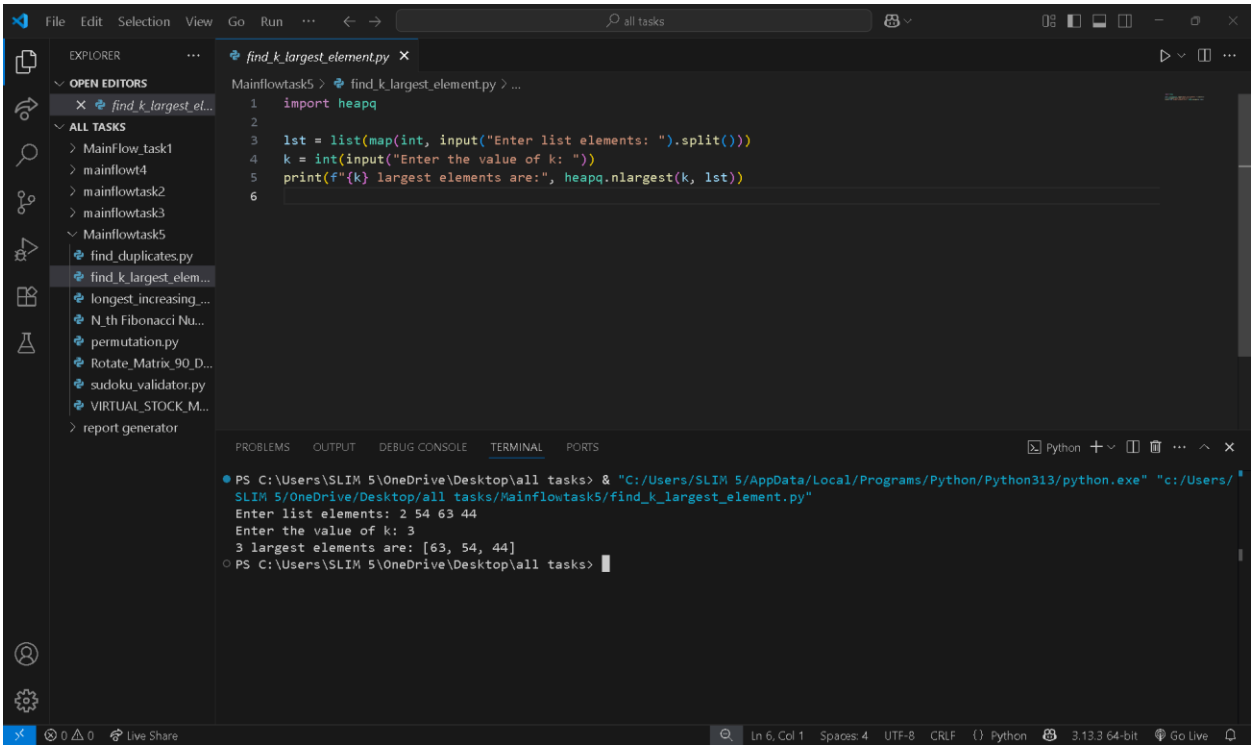
import heapq

def k_largest(nums, k):

return heapq.nlargest(k, nums)

print(k_largest([1, 23, 12, 9, 30, 2, 50], 3))

Screenshot:



Task 38: Rotate Matrix 90° Clockwise

Objective:
Rotate a matrix 90 degrees clockwise.

Input:
2D list (matrix)

Output:
Rotated matrix

Explanation:
First transpose the matrix, then reverse each row.

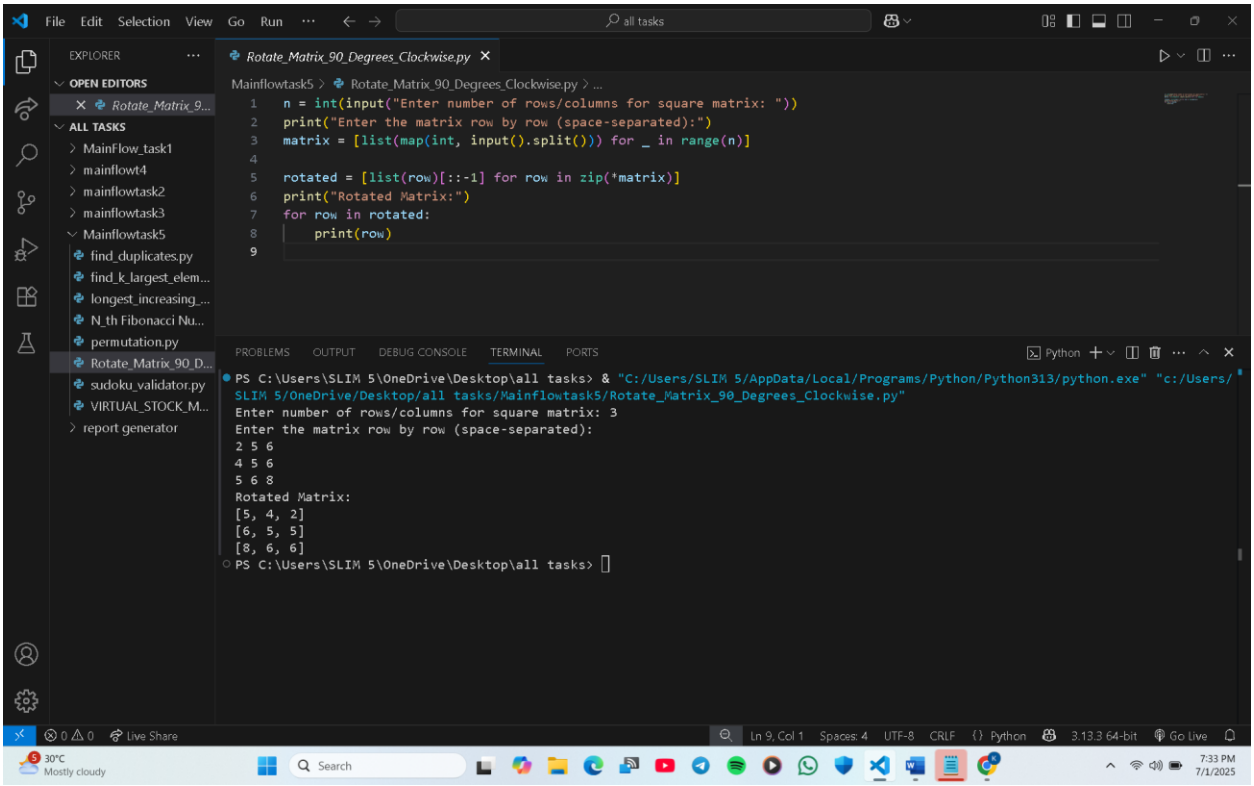
Python Code:

def rotate_matrix(matrix):

```
    return [list(reversed(col)) for col in zip(*matrix)]


matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
rotated = rotate_matrix(matrix)
for row in rotated:
    print(row)
```

Screenshot:



Task 39: Sudoku Validator

Objective:
Check if a Sudoku board is valid.

Input:
9x9 2D list (board)

Output:
True if valid, else False

Explanation:
We check all rows, columns, and 3×3 subgrids for duplicates.

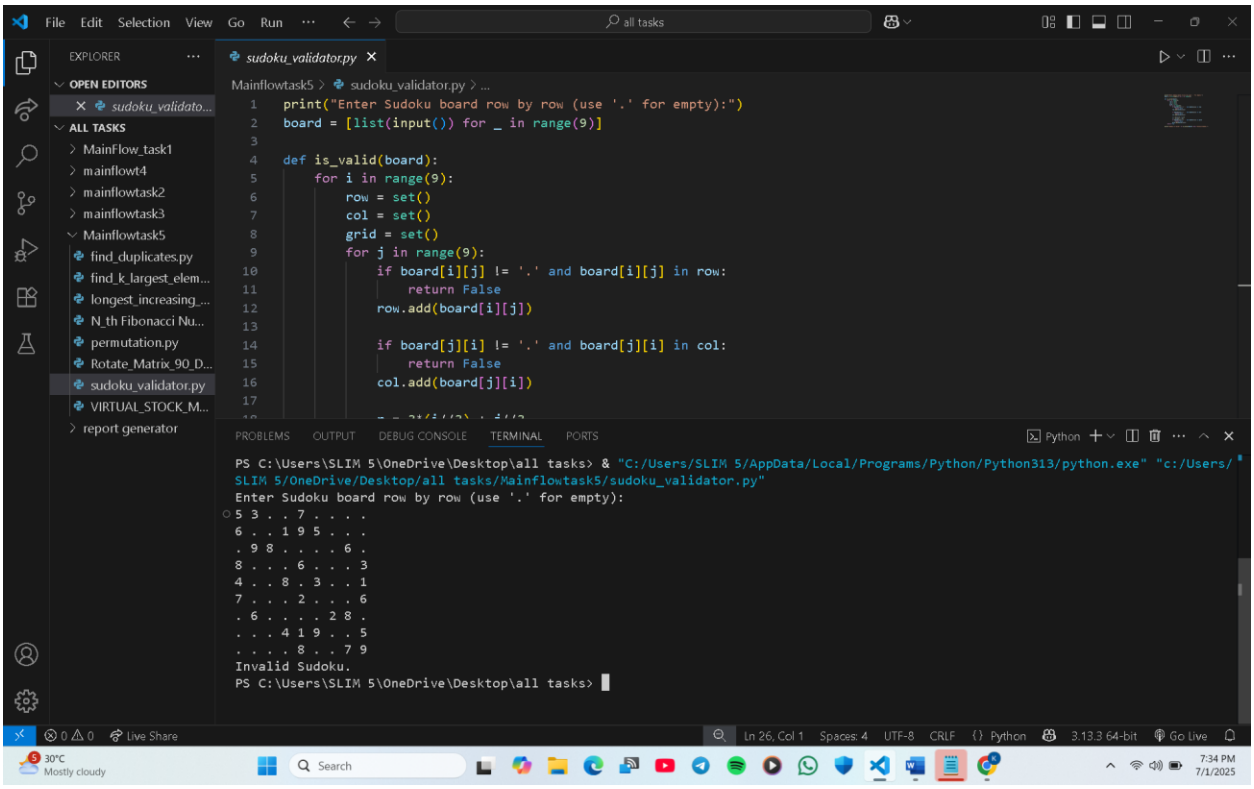
Python Code:

```
def is_valid_sudoku(board):
    def is_valid(block):
        nums = [x for x in block if x != "."]
        return len(set(nums)) == len(nums)

    for row in board:
        if not is_valid(row):
            return False

    for col in zip(*board):
        if not is_valid(col):
            return False

    for i in range(0,9,3):
        for j in range(0,9,3):
            block = [board[x][y] for x in range(i,i+3) for y in range(j,j+3)]
            if not is_valid(block):
                return False
    return True
```



Project: Virtual Stock Market Simulator

Description:
A simulation of a stock market where users can buy/sell virtual stocks with prices that fluctuate randomly.

Features:

- Users can buy/sell stocks.
- Prices change randomly or follow trends.
- Track user portfolios and transaction history.

Restrictions:

- No real-time API or market data is used.
- Prices are simulated using algorithms/random functions.

Challenges:

- Simulating realistic price fluctuations

- Designing clean business logic
- Managing portfolios

Learning Outcome:

- Students gain hands-on experience in:
    - Simulating complex systems
    - Financial modeling
    - Logic design without real data dependency

Python Code Snippet (Concept):

```
import random

stocks = {'AAPL': 100, 'GOOG': 150, 'TSLA': 200}
portfolio = {'cash': 10000, 'stocks': {}}

def simulate_price(stock):
    return round(stocks[stock] * (1 + random.uniform(-0.05, 0.05)), 2)

def buy(stock, quantity):
    global portfolio
    price = simulate_price(stock)
    cost = price * quantity
    if portfolio['cash'] >= cost:
        portfolio['cash'] -= cost
        portfolio['stocks'][stock] = portfolio['stocks'].get(stock, 0) + quantity
        print(f"Bought {quantity} shares of {stock} at {price}")
    else:
        print("Not enough cash.")
```

buy('AAPL', 10)

print(portfolio)

Screenshots