
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.

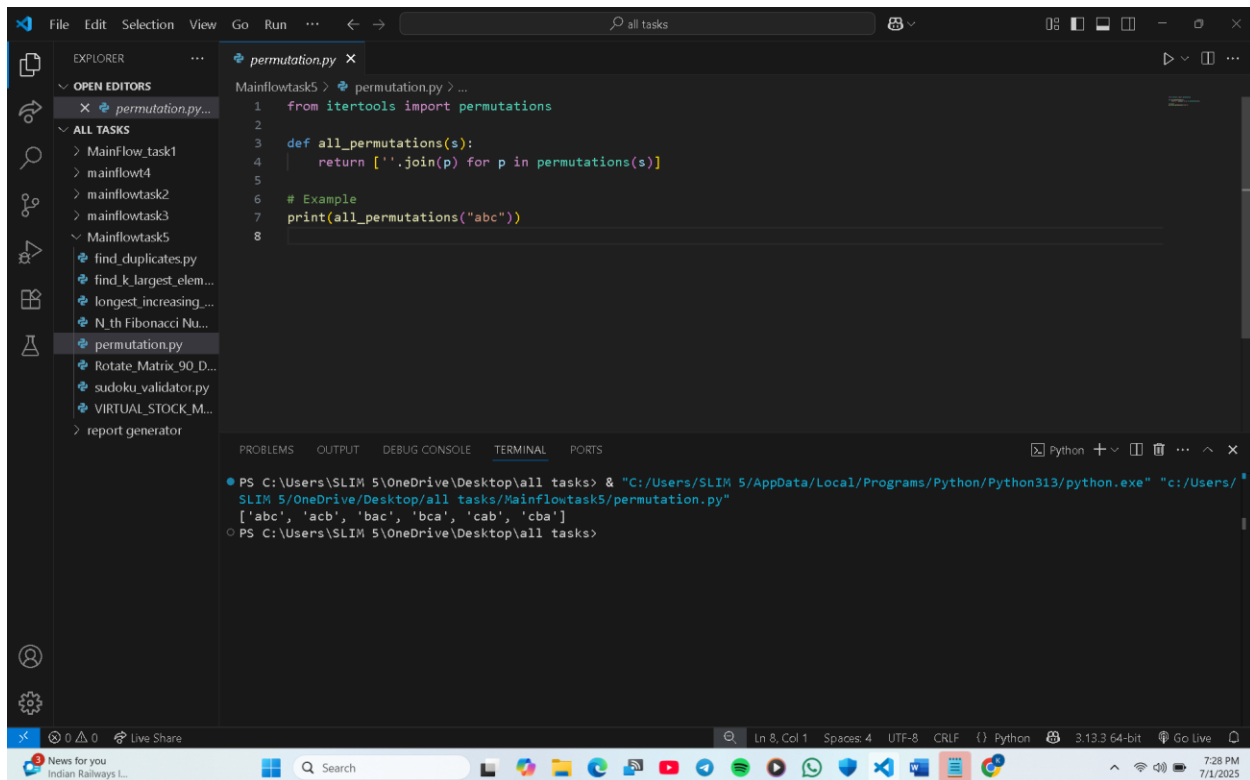
Python Code:

```
from itertools import permutations

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

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

Screenshot:



The screenshot shows a Visual Studio Code editor window. The Explorer sidebar on the left displays a project structure with folders 'MainFlow_task1' through 'MainFlowtask5' and a file 'permutation.py' under 'MainFlowtask5'. The Editor pane shows the content of 'permutation.py':

```
1 from itertools import permutations
2
3 def all_permutations(s):
4     return [''.join(p) for p in permutations(s)]
5
6 # Example
7 print(all_permutations("abc"))
8
```

The Terminal pane at the bottom shows the command prompt output:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/Mainflowtask5/permutation.py"
['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
```

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
```

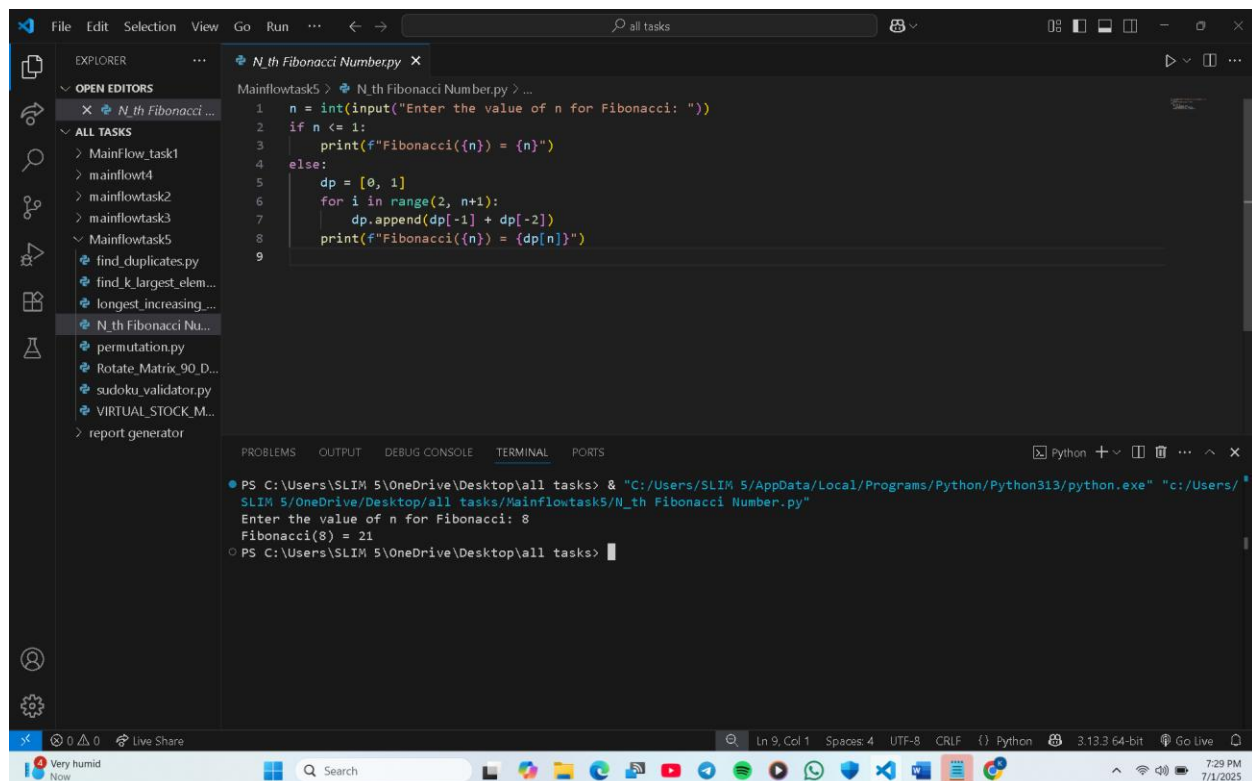
```
for i in range(2, n+1):
```

```
    fib[i] = fib[i-1] + fib[i-2]
```

```
return fib[n]
```

```
print(fibonacci(10))
```

Screenshot:



```
File Edit Selection View Go Run ... all tasks
EXPLORER
  OPEN EDITORS
    N_th Fibonacci Number.py
  ALL TASKS
    Mainflow_task1
    mainflowtask4
    mainflowtask2
    mainflowtask3
    Mainflowtask5
    find_duplicates.py
    find_k_largest_elem...
    longest_increasing_...
    N_th Fibonacci Nu...
    permutation.py
    Rotate_Matrix_90_D...
    sudoku_validator.py
    VIRTUAL_STOCK_M...
    report_generator
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
    Python
  PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/Mainflowtask5/N_th Fibonacci Number.py"
  Enter the value of n for Fibonacci: 8
  Fibonacci(8) = 21
  PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
```

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:

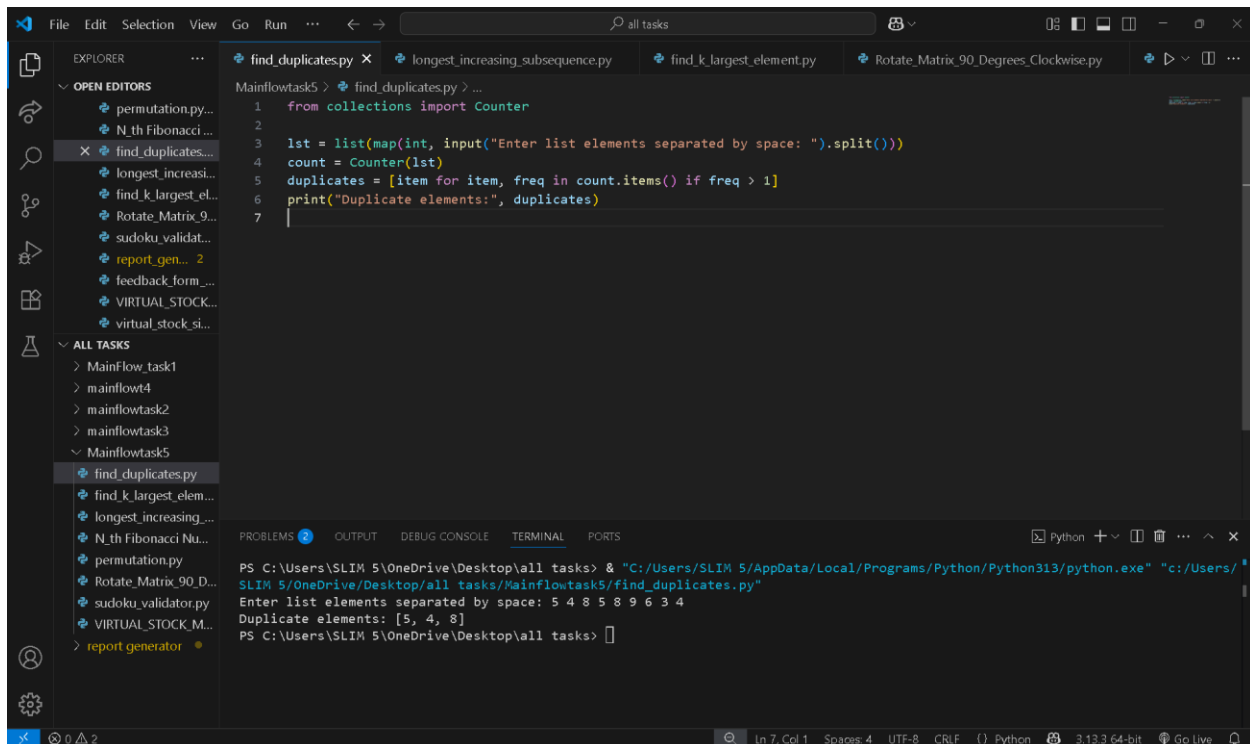
```
from collections import Counter
```

```
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]))
```



The screenshot shows a Visual Studio Code editor with a Python file named `find_duplicates.py` open. The code in the file is as follows:

```
1 from collections import Counter
2
3 lst = list(map(int, input("Enter list elements separated by space: ").split()))
4 count = Counter(lst)
5 duplicates = [item for item, freq in count.items() if freq > 1]
6 print("Duplicate elements:", duplicates)
7
```

The left sidebar shows the Explorer view with a list of files, including `find_duplicates.py`. The bottom panel shows the TERMINAL view with the following output:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/Mainflowtask5/find_duplicates.py"
Enter list elements separated by space: 5 4 8 5 8 9 6 3 4
Duplicate elements: [5, 4, 8]
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
```

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:

The screenshot shows a VS Code editor with a file named `longest_increasing_subsequence.py` open. The code implements a dynamic programming solution for finding the Longest Increasing Subsequence (LIS) in a list of integers. The terminal output shows the program being executed with the input `5 6 7 8 51 4`, resulting in an output of `Length of LIS: 5`.

```
1 nums = list(map(int, input("Enter list elements: ").split()))
2 if not nums:
3     print("Empty list.")
4 else:
5     dp = [1] * len(nums)
6     for i in range(1, len(nums)):
7         for j in range(i):
8             if nums[i] > nums[j]:
9                 dp[i] = max(dp[i], dp[j] + 1)
10    print("Length of LIS:", max(dp))
11
```

Terminal Output:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/Mainflowtask5/longest_increasing_subsequence.py"
Enter list elements: 5 6 7 8 51 4
Length of LIS: 5
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
```

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:

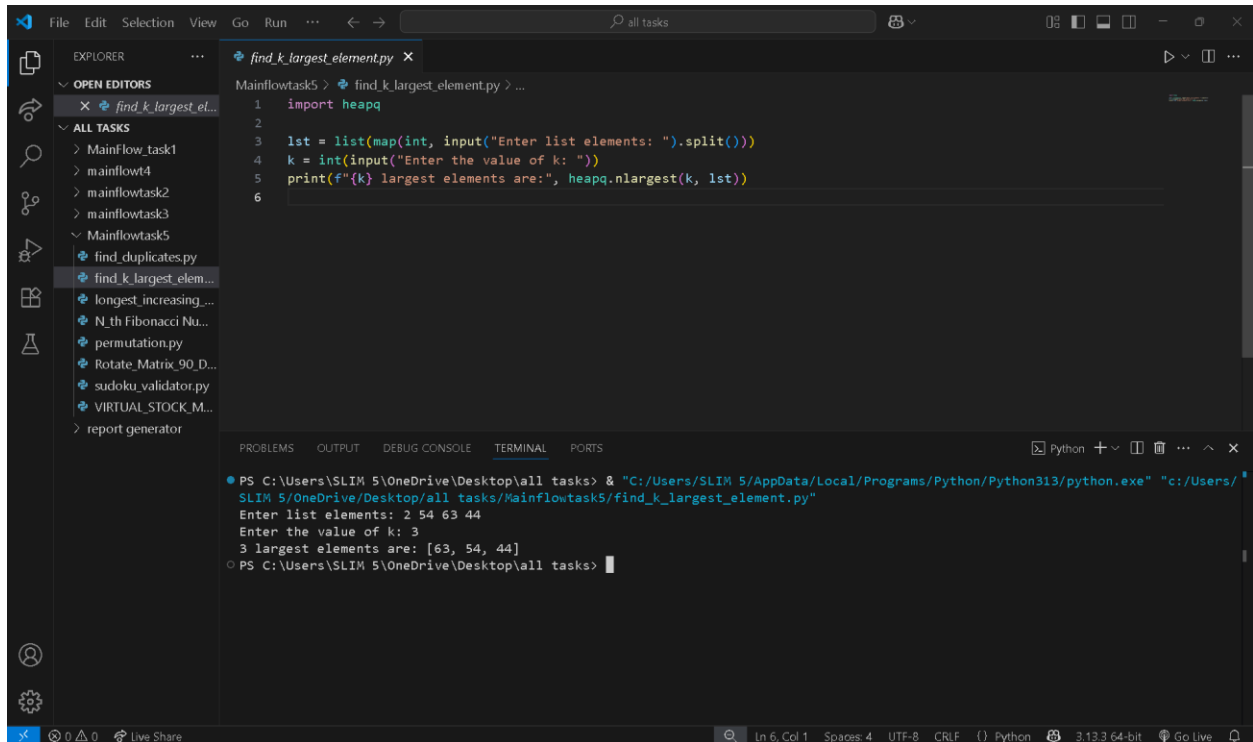
```
import heapq
```

```
def k_largest(nums, k):
```

```
return heapq.nlargest(k, nums)
```

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

Screenshot:



The screenshot shows a VS Code editor with a Python file named `find_k_largest_element.py` open. The code in the editor is as follows:

```
1 import heapq
2
3 lst = list(map(int, input("Enter list elements: ").split()))
4 k = int(input("Enter the value of k: "))
5 print(f"{k} largest elements are:", heapq.nlargest(k, lst))
6
```

The terminal output shows the execution of the script:

```
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/Mainflowtask5/find_k_largest_element.py"
Enter list elements: 2 54 63 44
Enter the value of k: 3
3 largest elements are: [63, 54, 44]
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
```

Task 38: Rotate Matrix 90° Clockwise

Objective:

Rotate a matrix 90 degrees clockwise.

Input:

2D list (matrix)

Output:

Rotated matrix

Explanation:

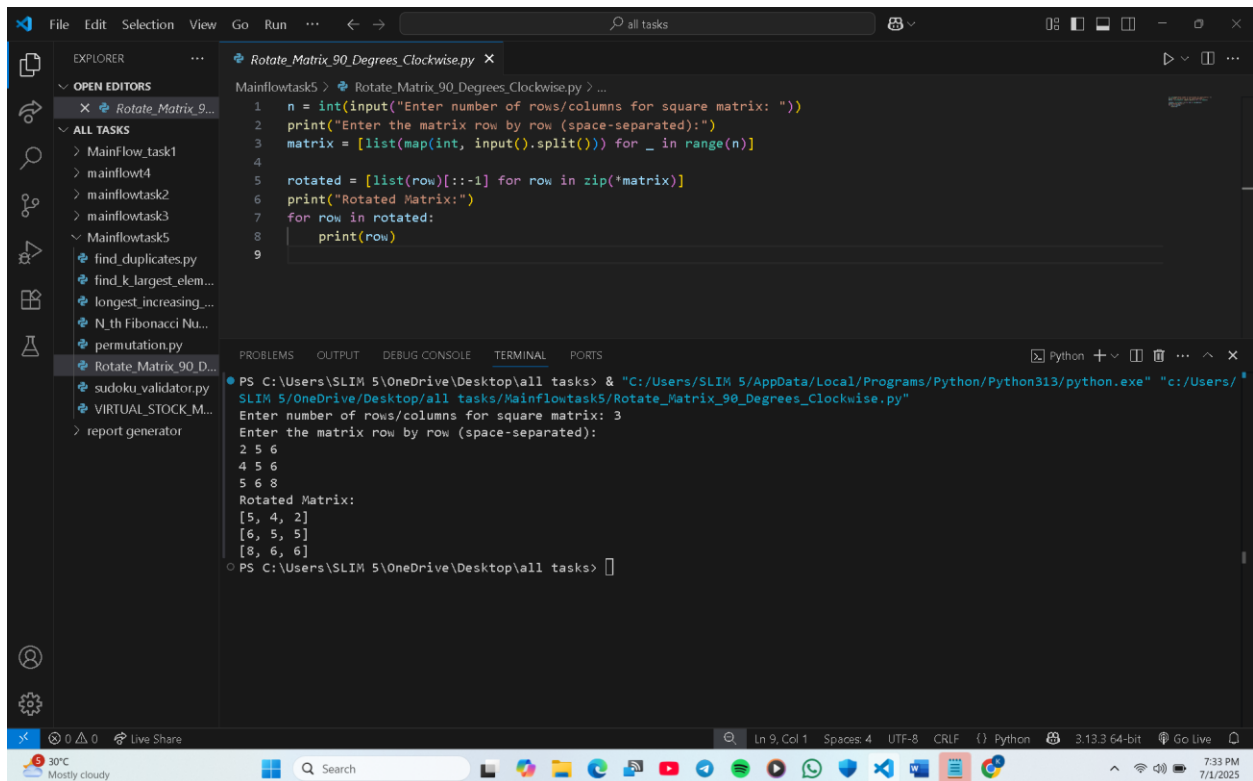
First transpose the matrix, then reverse each row.

Python Code:

```
def rotate_matrix(matrix):
```

```
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
```

The screenshot shows a Visual Studio Code editor window. The Explorer sidebar on the left displays a project structure with folders 'OPEN EDITORS' and 'ALL TASKS'. Under 'ALL TASKS', several files are listed, including 'MainFlow_task1', 'mainflowt4', 'mainflowtask2', 'mainflowtask3', 'Mainflowtask5', 'find_duplicates.py', 'find_k_largest_elem...', 'longest_increasing...', 'N_th Fibonacci Nu...', 'permutation.py', 'Rotate_Matrix_90_D...', 'sudoku_validator.py', 'VIRTUAL_STOCK.M...', and 'report generator'. The 'sudoku_validator.py' file is open in the editor. The code defines a function 'is_valid(board)' that checks if a 9x9 board is a valid Sudoku solution. It iterates through each row and column, using sets to ensure no duplicate numbers are present. The terminal at the bottom shows the command to run the script: 'PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/Mainflowtask5/sudoku_validator.py"'. The output shows the program prompting for a Sudoku board row by row, followed by an invalid board input and the message 'Invalid Sudoku.'

```
1 print("Enter Sudoku board row by row (use '.' for empty):")
2 board = [list(input()) for _ in range(9)]
3
4 def is_valid(board):
5     for i in range(9):
6         row = set()
7         col = set()
8         grid = set()
9         for j in range(9):
10            if board[i][j] != '.' and board[i][j] in row:
11                return False
12            row.add(board[i][j])
13            if board[j][i] != '.' and board[j][i] in col:
14                return False
15            col.add(board[j][i])
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

```
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks> & "C:/Users/SLIM 5/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SLIM 5/OneDrive/Desktop/all tasks/Mainflowtask5/sudoku_validator.py"
Enter Sudoku board row by row (use '.' for empty):
5 3 . . 7 . . . .
6 . . 1 9 5 . . .
. 9 8 . . . . 6 .
8 . . 6 . . . 3 .
4 . . 8 . 3 . . 1
7 . . 2 . . . 6 .
. 6 . . . . 2 8 .
. . . 4 1 9 . . 5
. . . 8 . . 7 9
Invalid Sudoku.
PS C:\Users\SLIM 5\OneDrive\Desktop\all tasks>
```

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

