```python
#33. Find All Permutations of a String
from itertools import permutations
def find_permutations(s):
    # Generate all permutations using itertools
    perm_list = [''.join(p) for p in permutations(s)]
    return perm_list
# Input
string = input("Enter a string: ")
result = find_permutations(string)
# Output
print("All permutations of the string are:")
print(result)
```

```
Enter a string:  main
```

```
All permutations of the string are:
['main', 'mani', 'mian', 'mina', 'mnai', 'mnia', 'amin', 'amni',
'aimn', 'ainm', 'anmi', 'anim', 'iman', 'imna', 'iamn', 'ianm',
'inma', 'inam', 'nmai', 'nmia', 'nami', 'naim', 'nima', 'niam']
```

```python
#34. N-th Fibonacci Number (Dynamic Programming)
def fibonacci(n):
    # Bottom-up approach with memoization array
    if n <= 0:
        return 0
    elif n == 1:
        return 1

    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]
# Input
n = int(input("Enter the value of n: "))
result = fibonacci(n)
# Output
print(f"The {n}-th Fibonacci number is: {result}")
```

```
Enter the value of n:  100
```

```
The 100-th Fibonacci number is: 354224848179261915075
```

```python
#35. Find Duplicates in a List
from collections import Counter
def find_duplicates(arr):
    # Count occurrences using Counter
    counter = Counter(arr)
    # Extract elements with count > 1
    duplicates = [item for item, count in counter.items() if count >
1]
    return duplicates
# Input
arr = list(map(int, input("Enter elements of the list separated by
space: ").split()))
result = find_duplicates(arr)
# Output
print("Duplicate elements in the list are:")
print(result)
```

```
Enter elements of the list separated by space:  2 2 5 7 9 9
```

```
Duplicate elements in the list are:
[2, 9]
```

```
#36. Longest Increasing Subsequence (LIS)
def longest_increasing_subsequence(arr):
    n = len(arr)
    lis = [1] * n  # Initialize LIS values for all indexes
    # Compute optimized LIS values
    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 the maximum value in lis[]
    return max(lis)
# Input
arr = list(map(int, input("Enter elements of the array separated by
space: ").split()))
result = longest_increasing_subsequence(arr)
# Output
print("Length of the longest increasing subsequence is:")
print(result)
```

```
Enter elements of the array separated by space:  1 2 3 4 5 0 1 5 7
```

```
Length of the longest increasing subsequence is:
6
```

```
#37. Find K Largest Elements
def find_k_largest(arr, k):
    # Sort the array in ascending order
    sorted_arr = sorted(arr)
    # Select the last k elements
    return sorted_arr[-k:]
# Input
arr = list(map(int, input("Enter elements of the list separated by
space: ").split()))
k = int(input("Enter the value of k: "))
# Ensure k is valid
```

```
if k > len(arr):
    print("Error: k cannot be greater than the number of elements in the list.")
else:
    result = find_k_largest(arr, k)
    # Output
    print(f"The {k} largest elements in the list are: {result}")
```

```
Enter elements of the list separated by space:  3 5 78 9 4
Enter the value of k:  3
```

```
The 3 largest elements in the list are: [5, 9, 78]
```

```
#38. Rotate Matrix
def rotate_matrix(matrix):
    # Transpose and then reverse each row for 90 degrees clockwise rotation
    return [list(reversed(col)) for col in zip(*matrix)]
# Input for n×n matrix
n = int(input("Enter the size of the square matrix (n x n): "))
print("Enter the matrix row by row (space-separated):")
matrix = []
for i in range(n):
    row = list(map(int, input().split()))
    if len(row) != n:
        print("Error: Each row must have exactly", n, "elements.")
        exit()
    matrix.append(row)
# Display Original Matrix
print("\nOriginal Matrix:")
for row in matrix:
    print(row)
# Rotate Matrix
rotated_matrix = rotate_matrix(matrix)
# Display Rotated Matrix
print("\nMatrix after 90 degrees clockwise rotation:")
for row in rotated_matrix:
    print(row)
```

```
Enter the size of the square matrix (n x n):  4
```

```
Enter the matrix row by row (space-separated):
```

```
 1 2 3 5
 3 4 5 7
 8 6 4 5
 2 5 6 7
```

```

Original Matrix:
[1, 2, 3, 5]
```

```
[3, 4, 5, 7]
[8, 6, 4, 5]
[2, 5, 6, 7]

Matrix after 90 degrees clockwise rotation:
[2, 8, 3, 1]
[5, 6, 4, 2]
[6, 4, 5, 3]
[7, 5, 7, 5]

#39. Sudoku Validator
def print_sudoku_board(board):
    print("\nSudoku Board:")
    for i, row in enumerate(board):
        # Add horizontal box separators
        if i % 3 == 0 and i != 0:
            print("-" * 21)

        for j, value in enumerate(row):
            # Add vertical box separators
            if j % 3 == 0 and j != 0:
                print("|", end=" ")
            print(value, end=" ")
        print()

def is_valid_sudoku(board):
    def is_valid_unit(unit):
        unit = [i for i in unit if i != '.']
        return len(unit) == len(set(unit))

    # Check rows, columns, and 3x3 grids
    for row in board:
        if not is_valid_unit(row):
            return False

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

    for i in (0, 3, 6):
        for j in (0, 3, 6):
            grid = [board[x][y] for x in range(i, i+3) for y in
range(j, j+3)]
            if not is_valid_unit(grid):
                return False

    return True

# Input
# Input with validation
```

```python
board = []
print("Enter the Sudoku board row by row (use '.' for empty cells):")
for i in range(9):
    row = input(f"Row {i+1}: ").split()
    if len(row) != 9:
        print("Error: Each row must have exactly 9 elements. Please re-enter the row.")
        exit()
    board.append(row)


# Display Board
print_sudoku_board(board)

# Validation Result
result = is_valid_sudoku(board)

if result:
    print("\nThe given Sudoku board is valid.")
else:
    print("\nThe given Sudoku board is invalid.")
```

```
Enter the Sudoku board row by row (use '.' for empty cells):
```

```
Row 1:  1 . . . . . . . .
Row 2:  2 . . . . . . 5 .
Row 3:  3 . . . . . . . 1
Row 4:  . . . . . . . . .
Row 5:  . . . . . . 3 . .
Row 6:  9 . . . . . . . .
Row 7:  . . . . . 2 . . .
Row 8:  . . 8 . . . . . .
Row 9:  . . . . . . . . .
```

```
Sudoku Board:
1 . . | . . . | . . .
2 . . | . . . | . 5 .
3 . . | . . . | . . 1
---------------------
. . . | . . . | . . .
. . . | . . . | 3 . .
9 . . | . . . | . . .
---------------------
. . . | . . 2 | . . .
. . 8 | . . . | . . .
. . . | . . . | . . .
```

```
The given Sudoku board is valid.
```

```
### 5.Virtual Stock Market Simulator(project-5)
import random

class VirtualStockMarket:
    def __init__(self):
        self.stocks = {"AAPL": 150, "GOOGL": 2800, "TSLA": 700,
"AMZN": 3400}
        self.portfolio = {}
        self.balance = 10000  # Starting balance

    def display_stocks(self):
        print("\nAvailable Stocks and Prices:")
        for stock, price in self.stocks.items():
            print(f"{stock}: ${price}")

    def update_prices(self):
        for stock in self.stocks:
            change = random.uniform(-0.05, 0.05)  # Price change
between -5% to +5%
            self.stocks[stock] = round(self.stocks[stock] * (1 +
change), 2)

    def buy_stock(self, symbol, quantity):
        if symbol in self.stocks:
            total_price = self.stocks[symbol] * quantity
            if total_price <= self.balance:
                self.balance -= total_price
                if symbol in self.portfolio:
                    self.portfolio[symbol] += quantity
                else:
                    self.portfolio[symbol] = quantity
                print(f"Bought {quantity} shares of {symbol}.")
            else:
                print("Insufficient balance.")
        else:
            print("Invalid stock symbol.")

    def sell_stock(self, symbol, quantity):
        if symbol in self.portfolio and self.portfolio[symbol] >=
quantity:
            total_price = self.stocks[symbol] * quantity
            self.balance += total_price
            self.portfolio[symbol] -= quantity
            if self.portfolio[symbol] == 0:
                del self.portfolio[symbol]
            print(f"Sold {quantity} shares of {symbol}.")
        else:
            print("Not enough shares to sell or invalid stock
symbol.")
```

```python
    def display_portfolio(self):
        print("\nYour Portfolio:")
        if not self.portfolio:
            print("No stocks owned.")
        for stock, qty in self.portfolio.items():
            value = self.stocks[stock] * qty
            print(f"{stock}: {qty} shares, Value: ${value:.2f}")
        print(f"Balance: ${self.balance:.2f}")

    def simulate(self):
        while True:
            self.update_prices()
            self.display_stocks()
            self.display_portfolio()

            print("\nOptions: ")
            print("1. Buy Stock")
            print("2. Sell Stock")
            print("3. Exit")

            choice = input("Enter your choice: ")
            if choice == '1':
                symbol = input("Enter stock symbol: ").upper()
                quantity = int(input("Enter quantity to buy: "))
                self.buy_stock(symbol, quantity)
            elif choice == '2':
                symbol = input("Enter stock symbol: ").upper()
                quantity = int(input("Enter quantity to sell: "))
                self.sell_stock(symbol, quantity)
            elif choice == '3':
                print("Exiting the Virtual Stock Market. Goodbye!")
                break
            else:
                print("Invalid choice. Please try again.")

# Run the simulator
market = VirtualStockMarket()
market.simulate()
```

```
Available Stocks and Prices:
AAPL: $145.54
GOOGL: $2855.66
TSLA: $701.53
AMZN: $3552.76

Your Portfolio:
No stocks owned.
Balance: $10000.00
```

```
Options:
1. Buy Stock
2. Sell Stock
3. Exit

Enter your choice:  1
Enter stock symbol:  AMZN
Enter quantity to buy:  2

Bought 2 shares of AMZN.

Available Stocks and Prices:
AAPL: $138.76
GOOGL: $2818.74
TSLA: $733.83
AMZN: $3608.91

Your Portfolio:
AMZN: 2 shares, Value: $7217.82
Balance: $2894.48

Options:
1. Buy Stock
2. Sell Stock
3. Exit

Enter your choice:  2
Enter stock symbol:  AMZN
Enter quantity to sell:  2

Sold 2 shares of AMZN.

Available Stocks and Prices:
AAPL: $138.27
GOOGL: $2922.06
TSLA: $730.13
AMZN: $3523.72

Your Portfolio:
No stocks owned.
Balance: $10112.30

Options:
1. Buy Stock
2. Sell Stock
3. Exit

Enter your choice:  3

Exiting the Virtual Stock Market. Goodbye!
```