```
#Task 33
def permute(s, path=""):
    if not s:
        print(path)  # Print the permutation
    else:
        for i in range(len(s)):
            permute(s[:i] + s[i+1:], path + s[i])

# Example Usage
s = "ABC"
permute(s)
```

```
ABC
ACB
BAC
BCA
CAB
CBA
```

```
#Task 34
def fibonacci(n):
    if n <= 1:
        return n

    dp = [0] * (n + 1)
    dp[1] = 1  # Base cases: F(0) = 0, F(1) = 1

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

    return dp[n]

# Example Usage
n = 10
print(fibonacci(n))  # Output: 55
```

```
55
```

```
#Task 35
def find_duplicates(lst):
    freq = {}  # Dictionary to store counts
    duplicates = set()

    for num in lst:
        if num in freq:
            duplicates.add(num)
        freq[num] = freq.get(num, 0) + 1

    return list(duplicates)

# Example Usage
nums = [1, 2, 3, 4, 5, 2, 3, 6, 7, 8, 1]
print(find_duplicates(nums))  # Output: [1, 2, 3]
```

```
[1, 2, 3]
```

```
#Task 36
def length_of_lis(nums):
    if not nums:
        return 0

    n = len(nums)
    dp = [1] * n  # Initialize LIS length for each element

    for i in range(n):
        for j in range(i):
            if nums[i] > nums[j]:  # Increasing sequence condition
                dp[i] = max(dp[i], dp[j] + 1)

    return max(dp)  # The longest LIS is the max value in dp[]

# Example Usage
nums = [10, 9, 2, 5, 3, 7, 101, 18]
print(length_of_lis(nums))  # Output: 4
```

```
4
```

```
#Task 37
import heapq

def k_largest_elements(nums, k):
    return heapq.nlargest(k, nums)  # Uses a min-heap internally

# Example Usage
nums = [3, 1, 5, 12, 2, 11, 7]
k = 3
print(k_largest_elements(nums, k))  # Output: [12, 11, 7]
```

```
[12, 11, 7]
```

```
#Task 38
def rotate_matrix(matrix):
    # Transpose: Convert rows to columns
    matrix = list(zip(*matrix))

    # Reverse each row
    return [list(row)[::-1] for row in matrix]

# Example Usage
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

rotated = rotate_matrix(matrix)
for row in rotated:
    print(row)
```

```
[7, 4, 1]
[8, 5, 2]
[9, 6, 3]
```

```
#Task 39
def is_valid_sudoku(board):
    seen = set()

    for i in range(9):
        for j in range(9):
            num = board[i][j]
            if num != ".":
                row_key = (i, num)          # Row identifier
                col_key = (num, j)          # Column identifier
                box_key = (i // 3, j // 3, num)  # 3x3 Box identifier

                if row_key in seen or col_key in seen or box_key in seen:
                    return False  # Duplicate found

                seen.update([row_key, col_key, box_key])  # Store unique numbers

    return True

# Example Usage
sudoku_board = [
    ["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"]
]

print(is_valid_sudoku(sudoku_board))
```

```
True
```

```
#Task 5
import random

class StockMarket:
```

```
        def __init__(self, stocks):
            self.stocks = {stock: random.uniform(50, 200) for stock in stocks}  # Random initial prices

        def update_prices(self):
            for stock in self.stocks:
                change = random.uniform(-5, 5)  # Simulating price fluctuations
                self.stocks[stock] = max(1, self.stocks[stock] + change)  # Ensure price doesn't drop below 1

        def get_price(self, stock):
            return self.stocks.get(stock, None)

    class Portfolio:
        def __init__(self):
            self.balance = 1000  # Starting cash
            self.holdings = {}

        def buy_stock(self, market, stock, quantity):
            price = market.get_price(stock)
            if price and self.balance >= price * quantity:
                self.balance -= price * quantity
                self.holdings[stock] = self.holdings.get(stock, 0) + quantity
                print(f"Bought {quantity} shares of {stock} at ${price:.2f}")
            else:
                print("Not enough balance or invalid stock.")

        def sell_stock(self, market, stock, quantity):
            price = market.get_price(stock)
            if stock in self.holdings and self.holdings[stock] >= quantity:
                self.holdings[stock] -= quantity
                self.balance += price * quantity
                print(f"Sold {quantity} shares of {stock} at ${price:.2f}")
                if self.holdings[stock] == 0:
                    del self.holdings[stock]
            else:
                print("Not enough stocks to sell.")

        def display_portfolio(self, market):
            print(f"\nBalance: ${self.balance:.2f}")
            print("Holdings:")
            for stock, qty in self.holdings.items():
                print(f"{stock}: {qty} shares @ ${market.get_price(stock):.2f} each")
            print("\n")

    # Sample Run
    def main():
        market = StockMarket(["AAPL", "GOOGL", "TSLA", "MSFT", "AMZN"])
        portfolio = Portfolio()

        while True:
            market.update_prices()
            print("\nStock Prices:")
            for stock, price in market.stocks.items():
                print(f"{stock}: ${price:.2f}")

            action = input("Choose action (buy/sell/view/exit): ").strip().lower()
            if action == "buy":
                stock = input("Enter stock symbol: ").strip().upper()
                qty = int(input("Enter quantity: "))
                portfolio.buy_stock(market, stock, qty)
            elif action == "sell":
                stock = input("Enter stock symbol: ").strip().upper()
                qty = int(input("Enter quantity: "))
                portfolio.sell_stock(market, stock, qty)
            elif action == "view":
                portfolio.display_portfolio(market)
            elif action == "exit":
                break
            else:
                print("Invalid action. Try again.")

    if __name__ == "__main__":
        main()
```

```
Stock Prices:
AAPL: $170.52
GOOGL: $182.26
TSLA: $172.18
MSFT: $62.07
AMZN: $86.64
Choose action (buy/sell/view/exit): buy
Enter stock symbol: AAPL
```

```
Enter quantity: 3
Bought 3 shares of AAPL at $170.52

Stock Prices:
AAPL: $173.19
GOOGL: $182.63
TSLA: $176.86
MSFT: $64.27
AMZN: $84.68
Choose action (buy/sell/view/exit): exit
```

Start coding or generate with AI.