```
def permutations_of_string(s):
    """
    Generates all permutations of a given string.

    Args:
        s: The input string.

    Returns:
        A list of all permutations.
    """
    if len(s) <= 1:
        return [s]

    result = []
    for i in range(len(s)):
        first_char = s[i]
        remaining_chars = s[:i] + s[i+1:]
        permutations_of_remaining = permutations_of_string(remaining_chars)
        for perm in permutations_of_remaining:
            result.append(first_char + perm)
    return result

def nth_fibonacci(n):
    """
    Finds the n-th Fibonacci number using dynamic programming.

    Args:
        n: The input integer.

    Returns:
        The n-th Fibonacci number.
    """
    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]

def find_duplicates(nums):
    """
    Identifies all duplicate elements in a list.

    Args:
        nums: The input list of integers.

    Returns:
        A list of duplicate integers.
    """
    counts = {}
    duplicates = []

    for num in nums:
        counts[num] = counts.get(num, 0) + 1

    for num, count in counts.items():
        if count > 1:
            duplicates.append(num)
    return duplicates

def longest_increasing_subsequence(nums):
    """
    Finds the length of the longest increasing subsequence in an array.

    Args:
        nums: The input list of integers.

    Returns:
        The length of the LIS.
    """
    if not nums:
        return 0

    n = len(nums)
    dp = [1] * n

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

    return max(dp)

def find_k_largest(nums, k):
    """
    Finds the k largest elements in a list.

    Args:
        nums: The input list of integers.
        k: The number of largest elements to find.

    Returns:
        A list of the k largest integers.
    """
    nums.sort(reverse=True)
    return nums[:k]
# Example usage:
print(permutations_of_string("abc"))
print(nth_fibonacci(10))
print(find_duplicates([1, 2, 3, 4, 2, 5, 3]))
print(longest_increasing_subsequence([10, 9, 2, 5, 3, 7, 101, 18]))
print(find_k_largest([3, 1, 4, 1, 5, 9, 2, 6], 3))

import random

def rotate_matrix(matrix):
    """
    Rotates a matrix 90 degrees clockwise.

    Args:
        matrix: A 2D list (matrix).

    Returns:
        The rotated matrix.
    """
    rows = len(matrix)
    cols = len(matrix[0])

    rotated = [[0] * rows for _ in range(cols)]
    for i in range(rows):
        for j in range(cols):
            rotated[j][rows - 1 - i] = matrix[i][j]

    return rotated

def is_valid_sudoku(board):
    """
    Validates whether a given Sudoku board configuration is valid.

    Args:
        board: A 9x9 2D list representing a Sudoku board.

    Returns:
        True if valid, otherwise False.
    """
    def is_valid(arr):
        seen = set()
        for val in arr:
            if val != '.':
                if val in seen:
                    return False
                seen.add(val)
        return True

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

    for col in range(9):
        if not is_valid([board[row][col] for row in range(9)]):
            return False

    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            subgrid = [board[row][col] for row in range(i, i + 3) for col in range(j, j + 3)]
            if not is_valid(subgrid):
                return False

    return True

class StockMarketSimulator:
    def __init__(self, initial_prices):
        self.stocks = {stock: price for stock, price in initial_prices.items()}
        self.portfolios = {}
        self.transactions = []

    def simulate_price_change(self):
        for stock in self.stocks:
            change = random.uniform(-0.1, 0.1)  # Random fluctuation (-10% to +10%)
            self.stocks[stock] = max(0.1, self.stocks[stock] * (1 + change))  # Ensure price stays positive

    def buy_stock(self, user, stock, quantity):
        if stock not in self.stocks:
            return "Stock not available."

        price = self.stocks[stock]
        total_cost = price * quantity

        if user not in self.portfolios:
            self.portfolios[user] = {}
        if stock not in self.portfolios[user]:
            self.portfolios[user][stock] = 0

        self.portfolios[user][stock] += quantity
        self.transactions.append((user, "buy", stock, quantity, price))
        return f"{user} bought {quantity} shares of {stock} at ${price:.2f} per share."

    def sell_stock(self, user, stock, quantity):
        if stock not in self.stocks:
            return "Stock not available."

        if user not in self.portfolios or stock not in self.portfolios[user] or self.portfolios[user][stock] < quantity:
            return "Insufficient shares."

        price = self.stocks[stock]
        self.portfolios[user][stock] -= quantity
        self.transactions.append((user, "sell", stock, quantity, price))
        return f"{user} sold {quantity} shares of {stock} at ${price:.2f} per share."

    def get_portfolio(self, user):
        return self.portfolios.get(user, {})

    def get_stock_prices(self):
        return self.stocks

    def get_transactions(self):
        return self.transactions

# Example Usage:
```

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(rotate_matrix(matrix))

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))

initial_prices = {"AAPL": 150.0, "GOOG": 2700.0, "TSLA": 700.0}
simulator = StockMarketSimulator(initial_prices)

simulator.buy_stock("Alice", "AAPL", 10)
simulator.simulate_price_change()
print(simulator.get_stock_prices())
simulator.sell_stock("Alice", "AAPL", 5)
print(simulator.get_portfolio("Alice"))
print(simulator.get_transactions())
```