```
def permute(s, answer=""):
    if len(s) == 0:
        print(answer)
        return
    for i in range(len(s)):
        ch = s[i]
        left_substr = s[:i]
        right_substr = s[i+1:]
        rest = left_substr + right_substr
        permute(rest, answer + ch)

# Example usage
s = "ABC"
print("Permutations using recursion:")
permute(s)
```

```
def fibonacci(n):
    if n <= 0:
        return "Invalid input"
    elif n == 1 or n == 2:
        return 1

    fib = [0] * (n + 1)  # Memoization array
    fib[1], fib[2] = 1, 1  # Base cases

    for i in range(3, n + 1):
        fib[i] = fib[i - 1] + fib[i - 2]  # Bottom-up computation
```

```
    return fib[n]

# Example usage
n = 10
print(f"The {n}-th Fibonacci number is: {fibonacci(n)}")




def find_duplicates(lst):
    counts = {}  # Dictionary to store occurrences
    duplicates = []

    for num in lst:
        counts[num] = counts.get(num, 0) + 1  # Increment count

    for key, value in counts.items():
        if value > 1:  # If count is more than 1, it's a duplicate
            duplicates.append(key)

    return duplicates

# Example usage
lst = [1, 2, 3, 4, 5, 2, 3, 6, 7, 8, 1]
print("Duplicates:", find_duplicates(lst))
```

```
def length_of_lis(nums):
    if not nums:
        return 0

    dp = [1] * len(nums)  # Initialize DP array with 1

    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j] + 1)  # Update LIS length

    return max(dp)  # Max value in dp array is the LIS length

# Example usage
nums = [10, 9, 2, 5, 3, 7, 101, 18]
print("Length of LIS:", length_of_lis(nums))



def k_largest_elements_sort(nums, k):
    return sorted(nums, reverse=True)[:k]  # Sort in descending order and take first k elements

# Example usage
nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
k = 3
print("K largest elements (sorting):", k_largest_elements_sort(nums, k))



def rotate_matrix(matrix):
    n = len(matrix)
```

```
    # Step 1: Transpose the matrix (swap rows and columns)
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

    # Step 2: Reverse each row
    for row in matrix:
        row.reverse()

    return matrix

# Example usage
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
rotated_matrix = rotate_matrix(matrix)

print("Rotated Matrix:")
for row in rotated_matrix:
    print(row)




def is_valid_sudoku(board):
    def is_valid_unit(unit):
        nums = [num for num in unit if num != '.']  # Ignore empty cells
```

```
        return len(nums) == len(set(nums))  # Check for duplicates

    # Check rows
    for row in board:
        if not is_valid_unit(row):
            return False

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

    # Check 3×3 subgrids
    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            subgrid = [board[x][y] for x in range(i, i+3) for y in range(j, j+3)]
            if not is_valid_unit(subgrid):
                return False

    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 the Sudoku board valid?", is_valid_sudoku(sudoku_board))
```