DAY – 6

1.Remove Element

def removeElement(nums, val):

k = 0

for i in range(len(nums)):

if nums[i] != val:

nums[k] = nums[i]

k += 1

return k

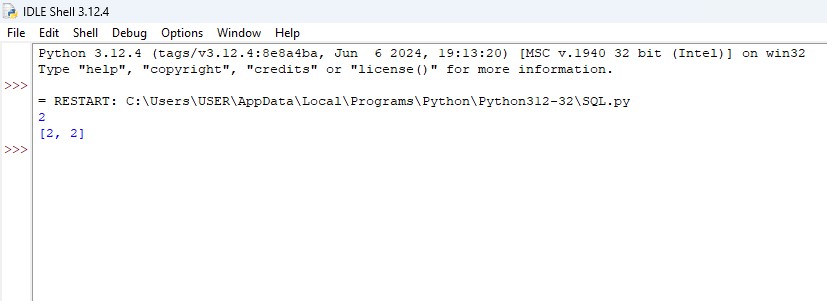
nums = [3, 2, 2, 3]

val = 3

k = removeElement(nums, val)

print(k)

print(nums[:k])



2. **Determine if a 9 x 9 Sudoku board is valid**

def isValidSudoku(board):

rows = [set() for \_ in range(9)]

cols = [set() for \_ in range(9)]

boxes = [set() for \_ in range(9)]

for r in range(9):

for c in range(9):

if board[r][c] == '.':

continue

num = board[r][c]

box\_index = (r // 3) \* 3 + (c // 3)

if num in rows[r] or num in cols[c] or num in boxes[box\_index]:

return False

rows[r].add(num)

cols[c].add(num)

boxes[box\_index].add(num)

return True

sudoku\_board = [ ["5","3",".",".","7",".",".",".","."],

["6",".",".","1","9","5",".",".","."],

[".","9","8",".",".",".",".","6","."],

["8",".",".",".","6",".",".",".","3"],

["4",".",".","8",".","3",".",".","1"],

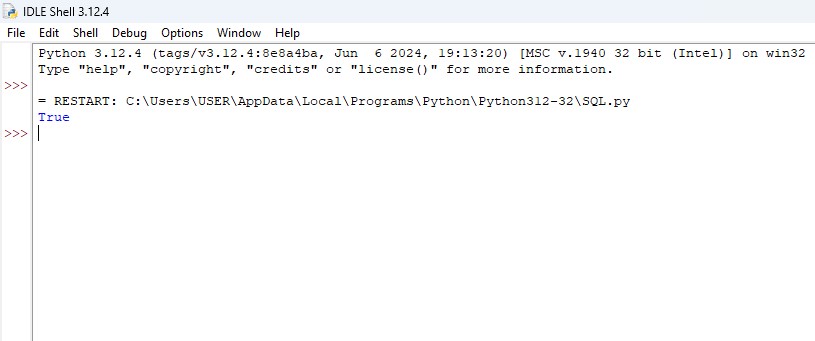
["7",".",".",".","2",".",".",".","6"],

[".","6",".",".",".",".","2","8","."],

[".",".",".","4","1","9",".",".","5"],

[".",".",".",".","8",".",".","7","9"]]

print(isValidSudoku(sudoku\_board))



3. Suduko Solver

def solveSudoku(board):

def isValid(board, r, c, num):

for i in range(9):

if board[r][i] == num:

return False

for i in range(9):

if board[i][c] == num:

return False

startRow, startCol = 3 \* (r // 3), 3 \* (c // 3)

for i in range(3):

for j in range(3):

if board[startRow + i][startCol + j] == num:

return False

return True

def solve(board):

for r in range(9):

for c in range(9):

if board[r][c] == '.':

for num in '123456789':

if isValid(board, r, c, num):

board[r][c] = num

if solve(board):

return True

board[r][c] = '.'

return False

return True

solve(board)

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","."],

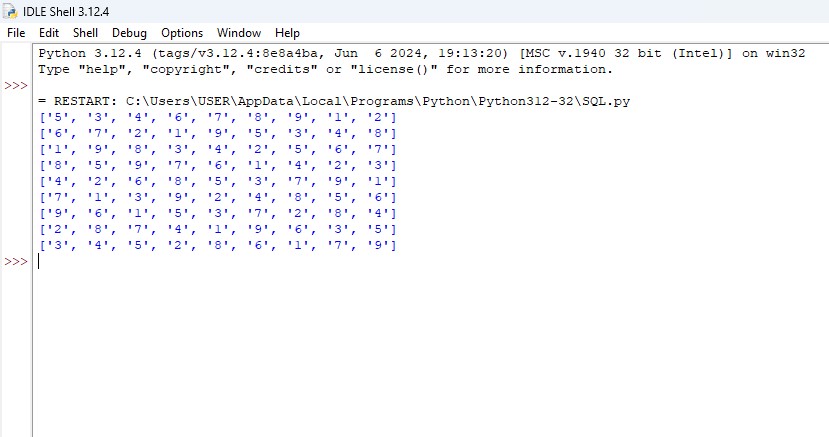
[".",".",".","4","1","9",".",".","5"],

[".",".",".",".","8",".",".","7","9"]]

solveSudoku(sudoku\_board)

for row in sudoku\_board:

print(row)



4. Count and Say

def countAndSay(n):

if n == 1:

return "1"

def next\_sequence(sequence):

result = []

i = 0

while i < len(sequence):

count = 1

while i + 1 < len(sequence) and sequence[i] == sequence[i + 1]:

i += 1

count += 1

result.append(str(count) + sequence[i])

i += 1

return ''.join(result)

current\_sequence = "1"

for \_ in range(1, n):

current\_sequence = next\_sequence(current\_sequence)

return current\_sequence

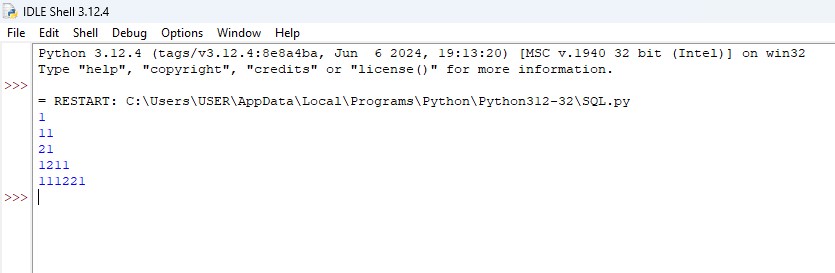
print(countAndSay(1))

print(countAndSay(2))

print(countAndSay(3))

print(countAndSay(4))

print(countAndSay(5))



5. Combination Sum

def combinationSum(candidates, target):

def backtrack(start, remaining, path):

if remaining == 0:

result.append(list(path))

return

elif remaining < 0:

return

for i in range(start, len(candidates)):

path.append(candidates[i])

backtrack(i, remaining - candidates[i], path)

path.pop()

candidates.sort()

result = []

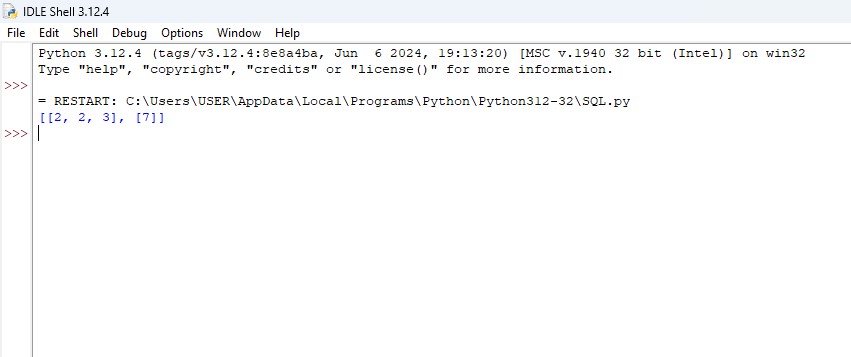
backtrack(0, target, [])

return result

candidates = [2, 3, 6, 7]

target = 7

print(combinationSum(candidates, target))



6. Combination Sum 11

def combinationSum2(candidates, target):

def backtrack(start, remaining, path):

if remaining == 0:

result.append(list(path))

return

elif remaining < 0:

return

for i in range(start, len(candidates)):

if i > start and candidates[i] == candidates[i - 1]:

continue

path.append(candidates[i])

backtrack(i + 1, remaining - candidates[i], path)

path.pop()

candidates.sort()

result = []

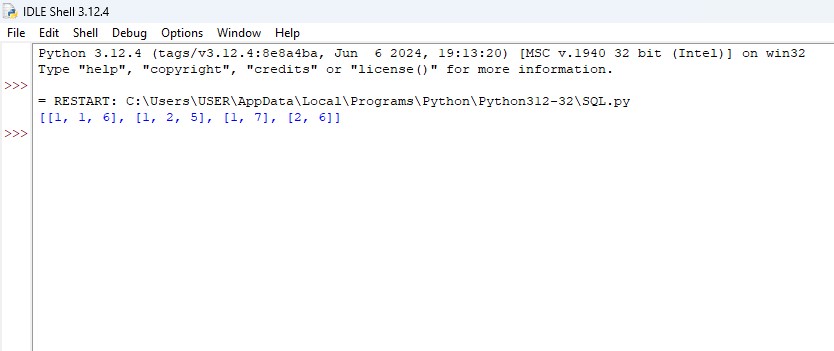
backtrack(0, target, [])

return result

candidates = [10, 1, 2, 7, 6, 1, 5]

target = 8

print(combinationSum2(candidates, target))



7. Permutations 11

def permuteUnique(nums):

def backtrack(path, used):

if len(path) == len(nums):

result.append(list(path))

return

for i in range(len(nums)):

if used[i] or (i > 0 and nums[i] == nums[i - 1] and not used[i - 1]):

continue

path.append(nums[i])

used[i] = True

backtrack(path, used)

path.pop()

used[i] = False

nums.sort()

result = []

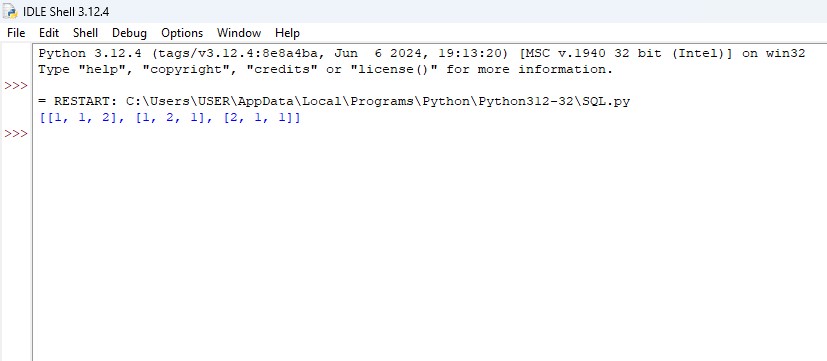
used = [False] \* len(nums)

backtrack([], used)

return result

nums = [1, 1, 2]

print(permuteUnique(nums))



8. Maximum Subarray

def maxSubArray(nums):

max\_current = max\_global = nums[0]

for num in nums[1:]:

max\_current = max(num, max\_current + num)

max\_global = max(max\_global, max\_current)

return max\_global

nums = [-2,1,-3,4,-1,2,1,-5,4]

print(maxSubArray(nums))

