1.Check If All 1's Are at Least Length K Places Away

def kLengthApart(nums, k):

pos = 0

count = 0

while (pos < len(nums) and nums[pos] == 0):

pos += 1

for i in range(pos + 1, len(nums)):

if nums[i] == 0:

count += 1

else :

if count < k:

return False

count = 0

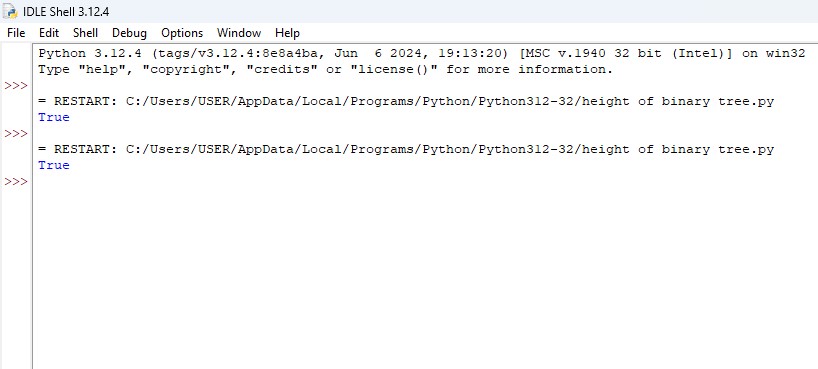
return True

if \_\_name\_\_ == "\_\_main\_\_":

nums = [ 1, 0, 0, 0, 1, 0, 0, 1, 0, 0 ]

k = 2

print(kLengthApart(nums, k))



2. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

from collections import deque

def longestSubarray(nums, limit):

max\_deque = deque()

min\_deque = deque()

left = 0

max\_length = 0

for right in range(len(nums)):

while max\_deque and nums[max\_deque[-1]] <= nums[right]:

max\_deque.pop()

max\_deque.append(right)

while min\_deque and nums[min\_deque[-1]] >= nums[right]:

min\_deque.pop()

min\_deque.append(right)

while nums[max\_deque[0]] - nums[min\_deque[0]] > limit:

left += 1

if max\_deque[0] < left:

max\_deque.popleft()

if min\_deque[0] < left:

min\_deque.popleft()

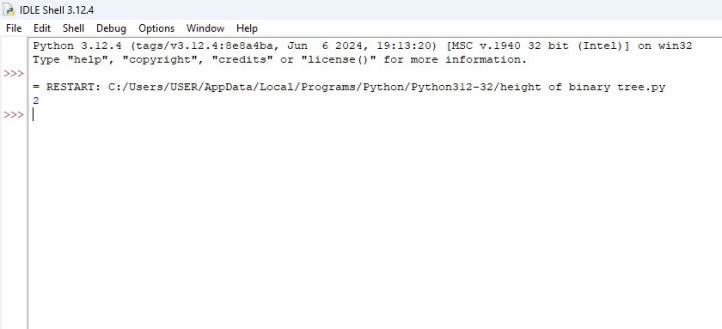
max\_length = max(max\_length, right - left + 1)

return max\_length

nums = [8,2,4,7]

limit = 2

print(longestSubarray(nums, limit))



3. Find the Kth Smallest Sum of a Matrix With Sorted Rows

import heapq

def kthSmallest(mat, k):

m, n = len(mat), len(mat[0])

min\_heap = [(sum(row[0] for row in mat), [0] \* m)]

visited = set(tuple([0] \* m))

for \_ in range(k):

curr\_sum, indices = heapq.heappop(min\_heap)

for i in range(m):

if indices[i] + 1 < n:

new\_indices = indices[:]

new\_indices[i] += 1

new\_tuple = tuple(new\_indices)

if new\_tuple not in visited:

new\_sum = curr\_sum - mat[i][indices[i]] + mat[i][new\_indices[i]]

heapq.heappush(min\_heap, (new\_sum, new\_indices))

visited.add(new\_tuple)

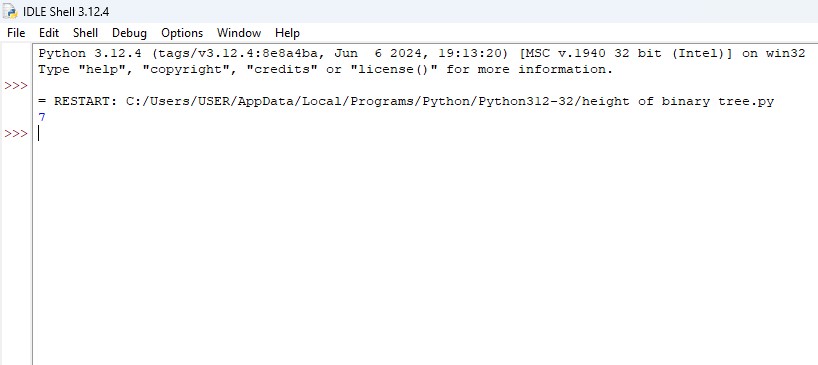
return curr\_sum

mat = [[1, 3, 11],

[2, 4, 6]]

k = 5

print(kthSmallest(mat, k))



4. Evaluate Boolean Expression

SQL Schema

import sqlite3

conn = sqlite3.connect(':memory:')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE Variables (

name TEXT PRIMARY KEY,

value INTEGER

)

''')

cursor.executemany('''

INSERT INTO Variables (name, value) VALUES (?, ?)

''', [('x', 66),

('y', 77),])

cursor.execute('''

CREATE TABLE Expressions (

left\_operand TEXT,

operator TEXT,

right\_operand TEXT,

PRIMARY KEY (left\_operand, operator, right\_operand)

)

''')

cursor.executemany('''

INSERT INTO Expressions (left\_operand, operator, right\_operand) VALUES (?, ?, ?)

''', [('x', '>', 'y'),

('y', '>', 'x'),

('y', '<', 'x'),

('x', '=', 'x'),

('x', '<', 'y'),

('x', '=', 'y')])

query = '''

SELECT

e.left\_operand,

e.operator,

e.right\_operand,

CASE

WHEN e.operator = '<' AND lv.value < rv.value THEN 'true'

WHEN e.operator = '>' AND lv.value > rv.value THEN 'true'

WHEN e.operator = '=' AND lv.value = rv.value THEN 'true'

ELSE 'false'

END AS result

FROM

Expressions e

JOIN

Variables lv ON e.left\_operand = lv.name

JOIN

Variables rv ON e.right\_operand = rv.name

'''

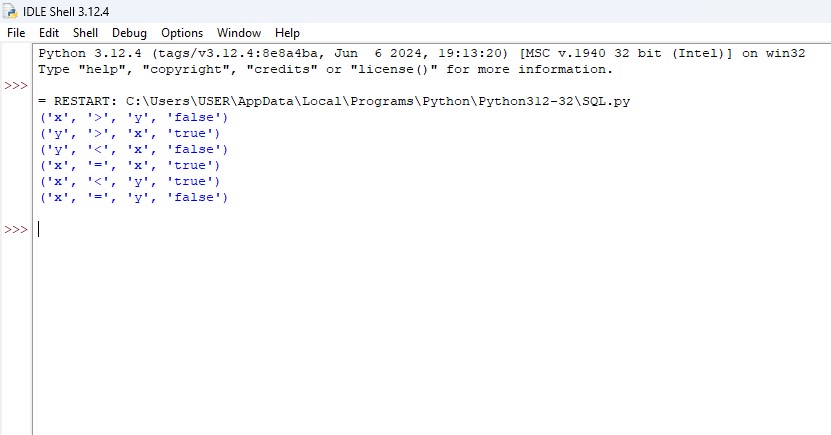
cursor.execute(query)

results = cursor.fetchall()

for row in results:

print(row)

conn.close()



5. Build an Array With Stack Operations

def buildArray(target, n):

result = []

current = 1

for num in target:

while current < num:

result.append("Push")

result.append("Pop")

current += 1

result.append("Push")

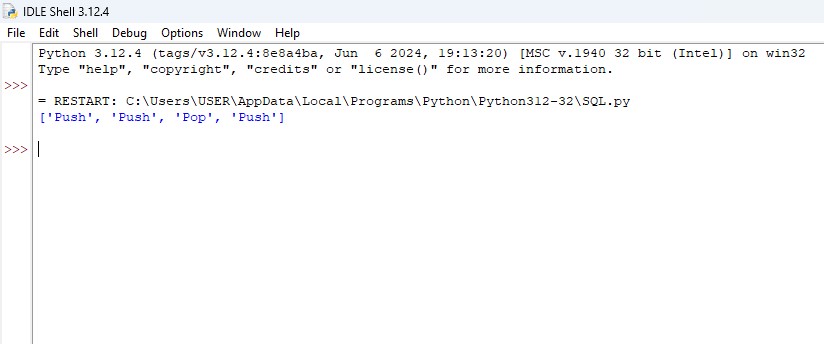
current += 1

return result

target = [1,3]

n = 3

print(buildArray(target, n))



6. Count Triplets That Can Form Two Arrays of Equal XOR

def countTriplets(arr):

n = len(arr)

prefix\_xor = [0] \* (n + 1)

for i in range(n):

prefix\_xor[i + 1] = prefix\_xor[i] ^ arr[i]

count = 0

for i in range(n):

for k in range(i + 1, n):

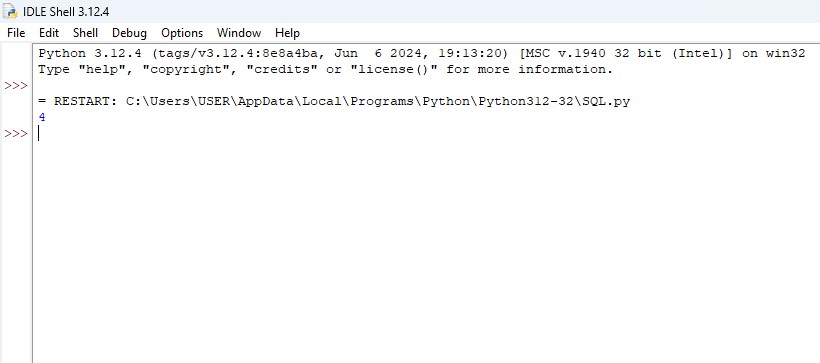
if prefix\_xor[i] == prefix\_xor[k + 1]:

count += k - i

return count

arr = [2, 3, 1, 6, 7]

print(countTriplets(arr))



7. . Minimum Time to Collect All Apples in a Tree

def minTime(n, edges, hasApple):

from collections import defaultdict

tree = defaultdict(list)

for u, v in edges:

tree[u].append(v)

tree[v].append(u)

def dfs(node, parent):

time = 0

for child in tree[node]:

if child != parent:

child\_time = dfs(child, node)

if child\_time > 0 or hasApple[child]:

time += child\_time + 2

return time

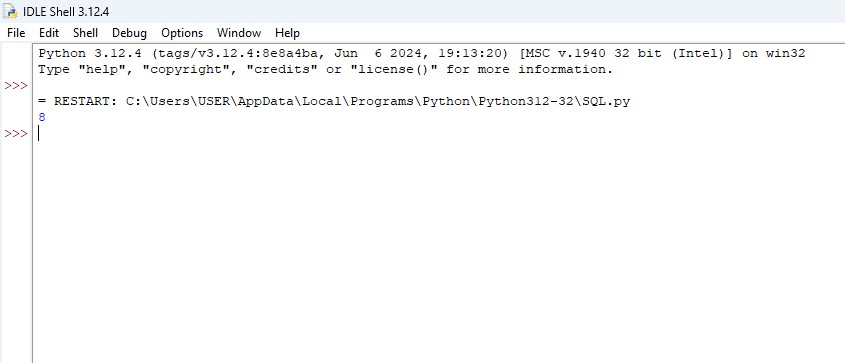
return dfs(0, -1)

n = 7

edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]]

hasApple = [False, False, True, False, True, True, False]

print(minTime(n, edges, hasApple))



8. Number of Ways of Cutting a Pizza

def numOfWays(pizza, k):

MOD = 10\*\*9 + 7

rows, cols = len(pizza), len(pizza[0])

prefix = [[0] \* (cols + 1) for \_ in range(rows + 1)]

for i in range(rows):

for j in range(cols):

prefix[i + 1][j + 1] = (prefix[i + 1][j] + prefix[i][j + 1] - prefix[i][j] + (1 if pizza[i][j] == 'A' else 0))

def hasApple(r1, c1, r2, c2):

return (prefix[r2 + 1][c2 + 1] - prefix[r2 + 1][c1] - prefix[r1][c2 + 1] + prefix[r1][c1]) > 0

dp = [[[0] \* (k + 1) for \_ in range(cols)] for \_\_ in range(rows)]

for i in range(rows):

for j in range(cols):

dp[i][j][1] = 1 if hasApple(i, j, rows - 1, cols - 1) else 0

for cut in range(2, k + 1):

for i in range(rows):

for j in range(cols):

for r in range(i + 1, rows):

if hasApple(i, j, r - 1, cols - 1):

dp[i][j][cut] = (dp[i][j][cut] + dp[r][j][cut - 1]) % MOD

for c in range(j + 1, cols):

if hasApple(i, j, rows - 1, c - 1):

dp[i][j][cut] = (dp[i][j][cut] + dp[i][c][cut - 1]) % MOD

return dp[0][0][k]

pizza = ["A..", "AAA", "..."]

k = 3

print(numOfWays(pizza, k))

