ASSIGNMENT (12.06.24)

1. Convert the Temperature

def convert\_temperature(celsius):

kelvin = celsius + 273.15

fahrenheit = celsius \* 1.80 + 32.00

return [round(kelvin, 5), round(fahrenheit, 5)]

celsius\_input\_1 = 36.50

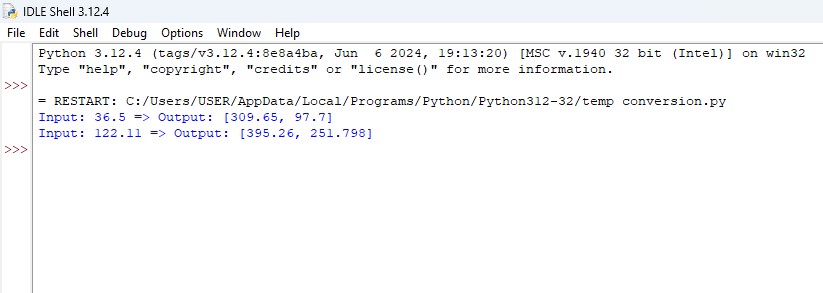
celsius\_input\_2 = 122.11

output\_1 = convert\_temperature(celsius\_input\_1)

output\_2 = convert\_temperature(celsius\_input\_2)

print(f"Input: {celsius\_input\_1} => Output: {output\_1}")

print(f"Input: {celsius\_input\_2} => Output: {output\_2}")



1. Number of Subarrays With LCM Equal to K

import math

from math import gcd

from functools import reduce

def lcm(a, b):

return abs(a \* b) // gcd(a, b)

def lcm\_list(numbers):

return reduce(lcm, numbers)

def count\_subarrays\_with\_lcm\_k(nums, k):

n = len(nums)

count = 0

for i in range(n):

current\_lcm = nums[i]

if current\_lcm == k:

count += 1

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

current\_lcm = lcm(current\_lcm, nums[j])

if current\_lcm == k:

count += 1

elif current\_lcm > k:

break

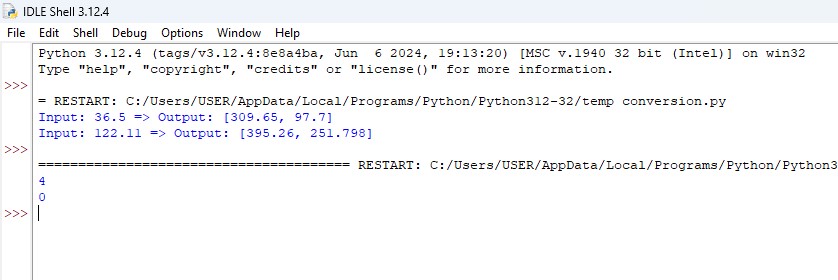
return count

nums1 = [3, 6, 2, 7, 1]

k1 = 6

print(count\_subarrays\_with\_lcm\_k(nums1, k1))

nums2 = [3]

k2 = 2print(count\_subarrays\_with\_lcm\_k(nums2, k2)) 

1. Minimum Number of Operations to Sort a Binary Tree by Level

from collections import deque

class TreeNode:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def minSwapsToSort(arr):

n = len(arr)

arr\_pos = list(enumerate(arr))

arr\_pos.sort(key=lambda it: it[1])

visited = [False] \* n

swaps = 0

for i in range(n):

if visited[i] or arr\_pos[i][0] == i:

continue

cycle\_size = 0

x = i

while not visited[x]:

visited[x] = True

x = arr\_pos[x][0]

cycle\_size += 1

if cycle\_size > 1:

swaps += cycle\_size - 1

return swaps

def minOperationsToSortTree(root):

if not root:

return 0

queue = deque([root])

operations = 0

while queue:

level\_size = len(queue)

level\_values = []

for \_ in range(level\_size):

node = queue.popleft()

level\_values.append(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

operations += minSwapsToSort(level\_values)

return operations

root = TreeNode(1)

root.left = TreeNode(4)

root.right = TreeNode(3)

root.left.left = TreeNode(7)

root.left.right = TreeNode(6)

root.right.left = TreeNode(8)

root.right.right = TreeNode(5)

root.left.left.left = None

root.left.left.right = None

root.left.right.left = None

root.left.right.right = None

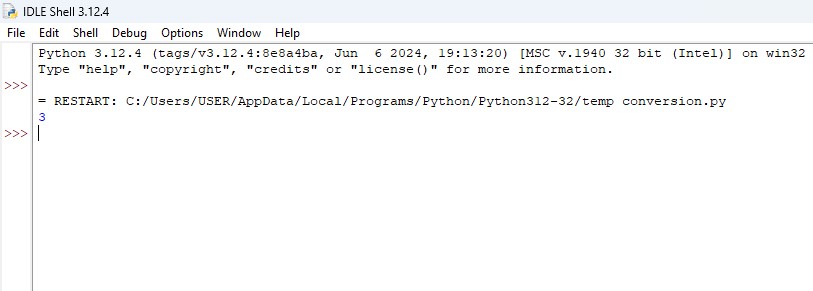
root.right.left.left = TreeNode(9)

root.right.left.right = None

root.right.right.left = TreeNode(10)

root.right.right.right = None

print(minOperationsToSortTree(root))



1. Maximum Number of Non-overlapping Palindrome Substrings

def maxPalindromicSubstrings(s, k):

n = len(s)

if k > n:

return 0

is\_palindrome = [[False] \* n for \_ in range(n)]

for i in range(n):

is\_palindrome[i][i] = True

for i in range(n - 1):

is\_palindrome[i][i + 1] = (s[i] == s[i + 1])

for length in range(3, n + 1):

for i in range(n - length + 1):

j = i + length - 1

is\_palindrome[i][j] = (s[i] == s[j]) and is\_palindrome[i + 1][j - 1]

count = 0

i = 0

while i <= n - k:

found = False

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

if is\_palindrome[i][j]:

count += 1

i = j + 1

found = True

break

if not found:

i += 1

return count

s1 = "abaccdbbd"

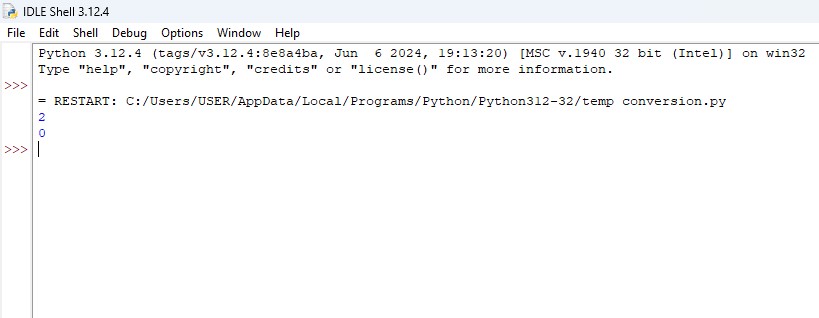
k1 = 3

print(maxPalindromicSubstrings(s1, k1))

s2 = "adbcda"

k2 = 2

print(maxPalindromicSubstrings(s2, k2))



1. Minimum Cost to Buy Apples

import heapq

def dijkstra(n, graph, start):

dist = [float('inf')] \* (n + 1)

dist[start] = 0

priority\_queue = [(0, start)]

while priority\_queue:

current\_dist, node = heapq.heappop(priority\_queue)

if current\_dist > dist[node]:

continue

for neighbor, weight in graph[node]:

distance = current\_dist + weight

if distance < dist[neighbor]:

dist[neighbor] = distance

heapq.heappush(priority\_queue, (distance, neighbor))

return dist

def minCostToBuyApples(n, roads, appleCost, k):

graph = [[] for \_ in range(n + 1)]

for a, b, cost in roads:

graph[a].append((b, cost))

graph[b].append((a, cost))

shortest\_paths = []

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

shortest\_paths.append(dijkstra(n, graph, i))

answer = []

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

min\_cost = float('inf')

for j in range(1, n + 1):

if i == j:

min\_cost = min(min\_cost, appleCost[j - 1])

else:

to\_j\_cost = shortest\_paths[i - 1][j]

from\_j\_cost = shortest\_paths[j - 1][i] \* k

total\_cost = to\_j\_cost + appleCost[j - 1] + from\_j\_cost

min\_cost = min(min\_cost, total\_cost)

answer.append(min\_cost)

return answer

n1 = 4

roads1 = [[1, 2, 4], [2, 3, 2], [2, 4, 5], [3, 4, 1], [1, 3, 4]]

appleCost1 = [56, 42, 102, 301]

k1 = 2

print(minCostToBuyApples(n1, roads1, appleCost1, k1))

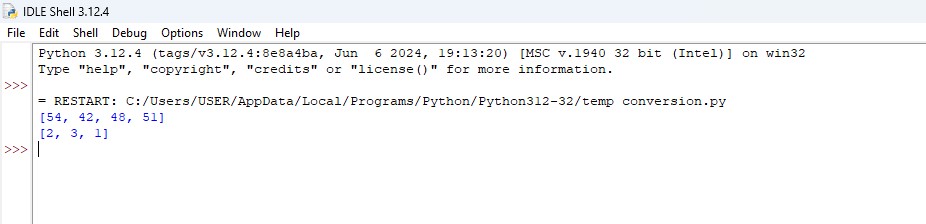
n2 = 3

roads2 = [[1, 2, 5], [2, 3, 1], [3, 1, 2]]

appleCost2 = [2, 3, 1]

k2 = 3

print(minCostToBuyApples(n2, roads2, appleCost2, k2))



1. Number of Unequal Triplets in Array

def countUnequalTriplets(nums):

n = len(nums)

count = 0

for i in range(n):

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

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

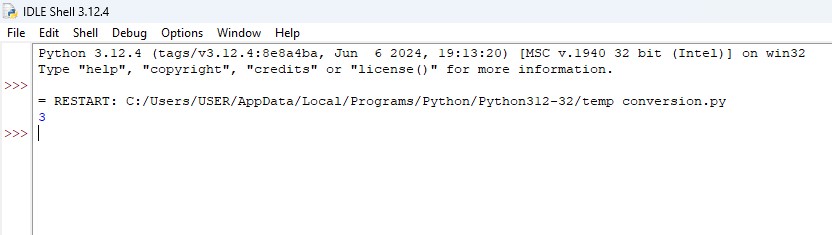
if nums[i] != nums[j] and nums[i] != nums[k] and nums[j] != nums[k]:

count += 1

return count

nums1 = [4, 4, 2, 4, 3]

print(countUnequalTriplets(nums1))



1. Closest Nodes Queries in a Binary Search Tree

from bisect import bisect\_left, bisect\_right

class TreeNode:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def inorder\_traversal(root):

result = []

stack = []

current = root

while stack or current:

while current:

stack.append(current)

current = current.left

current = stack.pop()

result.append(current.val)

current = current.right

return result

def closest\_nodes(root, queries):

sorted\_vals = inorder\_traversal(root)

result = []

for q in queries:

pos\_left = bisect\_right(sorted\_vals, q) - 1

pos\_right = bisect\_left(sorted\_vals, q)

mini = sorted\_vals[pos\_left] if pos\_left >= 0 else -1

maxi = sorted\_vals[pos\_right] if pos\_right < len(sorted\_vals) else -1

result.append([mini, maxi])

return result

root = TreeNode(6,

TreeNode(2,

TreeNode(1),

TreeNode(4)),

TreeNode(13,

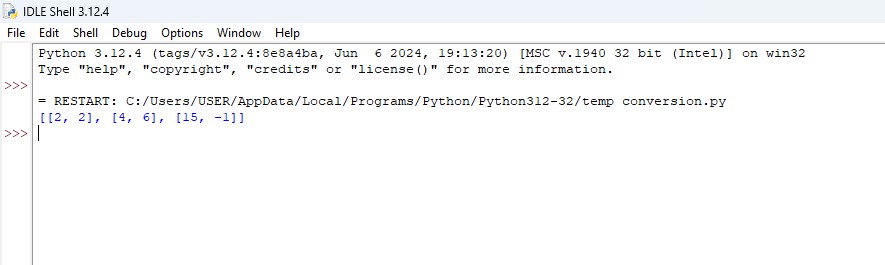
TreeNode(9),

TreeNode(15,

TreeNode(14))))

queries = [2, 5, 16]

print(closest\_nodes(root, queries))



1. Minimum Fuel Cost to Report to the Capital

from collections import defaultdict

import math

def minimumFuelCost(roads, seats):

graph = defaultdict(list)

for a, b in roads:

graph[a].append(b)

graph[b].append(a)

def dfs(node, parent):

total\_reps = 1

total\_fuel = 0

for neighbor in graph[node]:

if neighbor == parent:

continue

reps\_from\_subtree, fuel\_from\_subtree = dfs(neighbor, node)

total\_reps += reps\_from\_subtree

total\_fuel += fuel\_from\_subtree

if node != 0:

trips = math.ceil(total\_reps / seats)

total\_fuel += trips

return total\_reps, total\_fuel

\_, total\_fuel = dfs(0, -1)

return total\_fuel

roads1 = [[0, 1], [0, 2], [0, 3]]

seats1 = 5

print(minimumFuelCost(roads1, seats1))

roads2 = [[3, 1], [3, 2], [1, 0], [0, 4], [0, 5], [4, 6]]

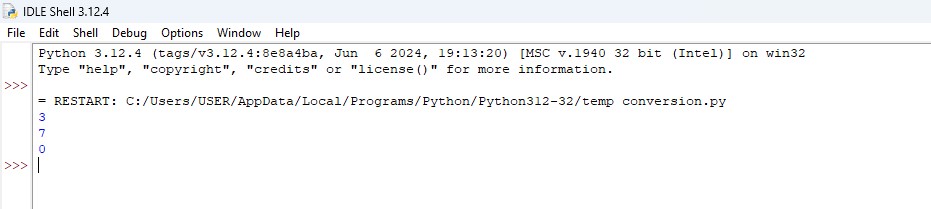
seats2 = 2

print(minimumFuelCost(roads2, seats2))

roads3 = []

seats3 = 1

print(minimumFuelCost(roads3, seats3))



1. Number of Beautiful Partitions

def is\_prime\_digit(ch):

return ch in {'2', '3', '5', '7'}

def is\_non\_prime\_digit(ch):

return ch in {'1', '4', '6', '8', '9'}

def beautiful\_partitions(s, k, minLength):

MOD = 10\*\*9 + 7

n = len(s)

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

dp[0][0] = 1

for i in range(n):

for j in range(k):

if dp[i][j] > 0:

for l in range(i + minLength, n + 1):

if is\_prime\_digit(s[i]) and is\_non\_prime\_digit(s[l - 1]):

dp[l][j + 1] = (dp[l][j + 1] + dp[i][j]) % MOD

return dp[n][k]

s1 = "23542185131"

k1 = 3

minLength1 = 2

print(beautiful\_partitions(s1, k1, minLength1))

