DAY – 9

1.Dynamic Prgramming

memo = {}

def fibonacci(n):

if n in memo:

return memo[n]

if n <= 2:

return 1

else:

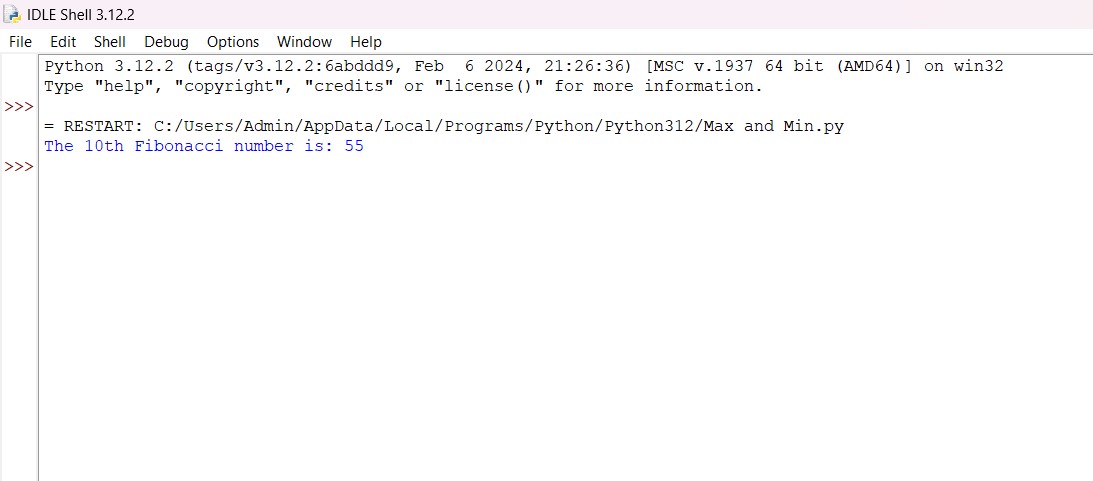
result = fibonacci(n-1) + fibonacci(n-2)

memo[n] = result

return result

n = 10

print(f"The {n}th Fibonacci number is:", fibonacci(n))



2. Dice Throw Problem

def dice\_throw(m, n, x):

"""

Function to calculate number of ways to get sum x with n dice each having m faces

:param m: number of faces on each dice (1 to m)

:param n: number of dice

:param x: desired sum

:return: number of ways to get sum x

"""

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

dp[0][0] = 1

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

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

dp[i][j] = 0

if j < i:

continue

for k in range(1, m + 1):

if j - k >= 0:

dp[i][j] += dp[i - 1][j - k]

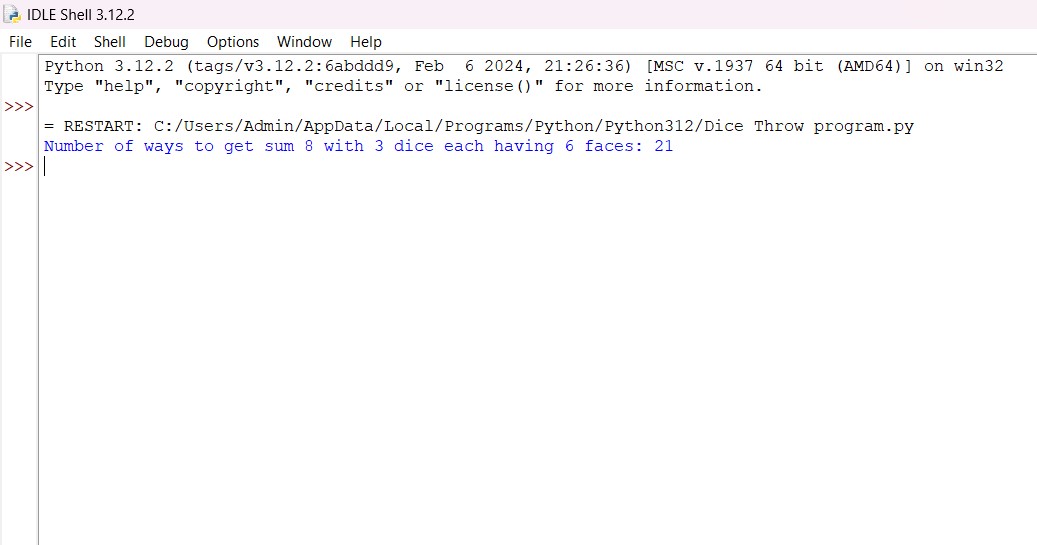
return dp[n][x]

m = 6

n = 3

x = 8

print(f"Number of ways to get sum {x} with {n} dice each having {m} faces: {dice\_throw(m, n, x)}")



3. Assembly line scheduling

def assembly\_line\_scheduling(a1, a2, t1, t2, e1, e2, x1, x2):

n = len(a1)

f1 = [0] \* n

f2 = [0] \* n

f1[0] = e1 + a1[0]

f2[0] = e2 + a2[0]

for i in range(1, n):

f1[i] = min(f1[i-1] + a1[i], f2[i-1] + t2[i-1] + a1[i])

f2[i] = min(f2[i-1] + a2[i], f1[i-1] + t1[i-1] + a2[i])

f1\_exit = f1[n-1] + x1

f2\_exit = f2[n-1] + x2

return min(f1\_exit, f2\_exit)

a1 = [7, 9, 3, 4, 8, 4]

a2 = [8, 5, 6, 4, 5, 7]

t1 = [2, 3, 1, 3, 4]

t2 = [4, 2, 1, 2, 2]

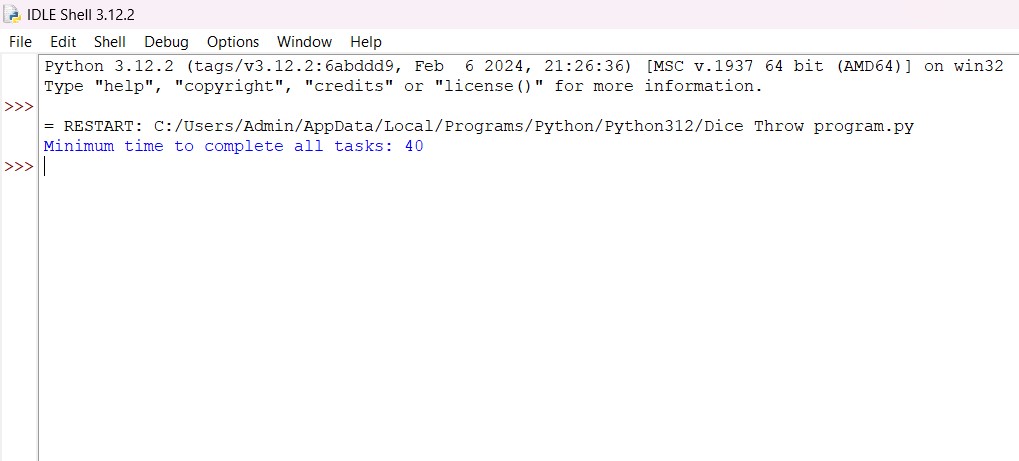
e1 = 2

e2 = 4

x1 = 3

x2 = 2

print("Minimum time to complete all tasks:", assembly\_line\_scheduling(a1, a2, t1, t2, e1, e2, x1, x2))



4. Travelling Salesman Problem

import sys

def tsp\_dp(graph):

n = len(graph)

dp = [[float('inf')] \* n for \_ in range(1 << n)]

dp[1][0] = 0

for mask in range(1 << n):

for u in range(n):

if mask & (1 << u):

for v in range(n):

if v != u and not (mask & (1 << v)) and graph[u][v] != float('inf'):

dp[mask | (1 << v)][v] = min(dp[mask | (1 << v)][v], dp[mask][u] + graph[u][v])

min\_cost = float('inf')

for v in range(1, n):

if graph[v][0] != float('inf'):

min\_cost = min(min\_cost, dp[(1 << n) - 1][v] + graph[v][0])

return min\_cost

graph = [[0, 10, 15, 20],

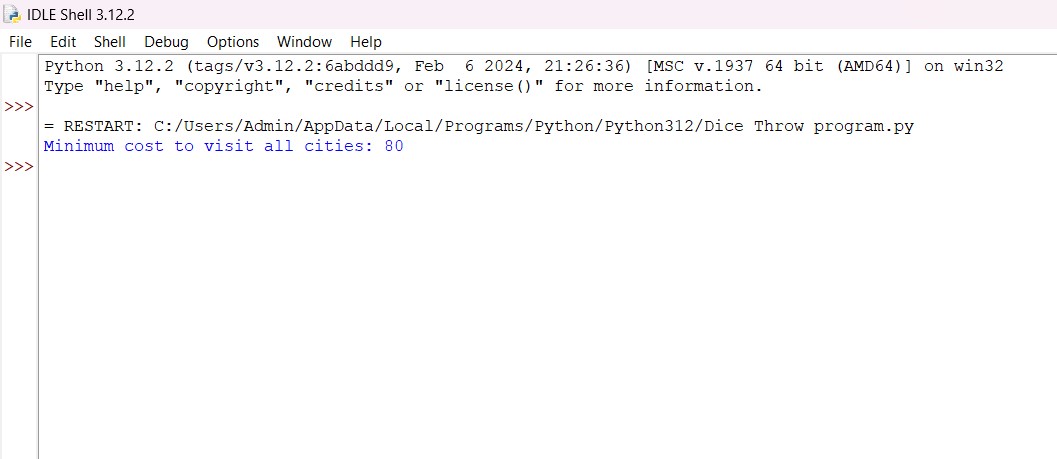
[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]]

min\_cost = tsp\_dp(graph)

print(f"Minimum cost to visit all cities: {min\_cost}")



5. Longest Palindrome Substring

def longest\_palindromic\_subsequence(s):

n = len(s)

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

for i in range(n):

dp[i][i] = 1

for L in range(2, n+1):

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

j = i + L - 1

if s[i] == s[j] and L == 2:

dp[i][j] = 2

elif s[i] == s[j]:

dp[i][j] = dp[i+1][j-1] + 2

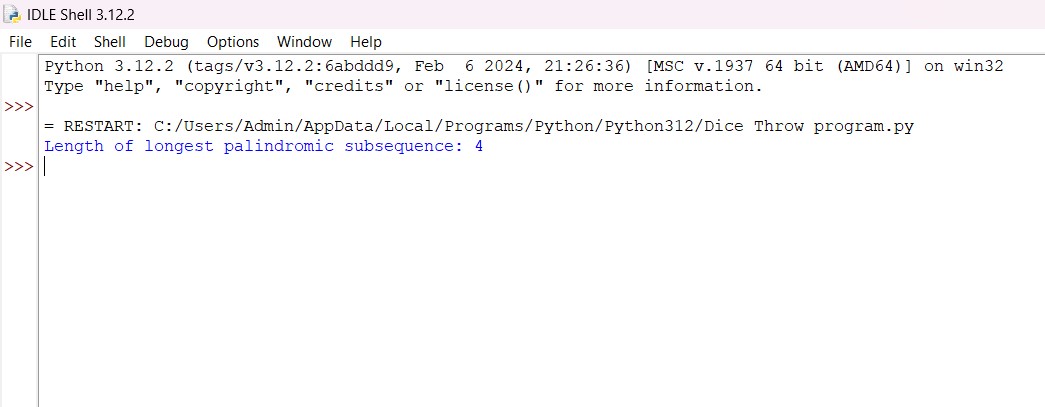
else:

dp[i][j] = max(dp[i][j-1], dp[i+1][j])

return dp[0][n-1]

s = "bbbab"

print("Length of longest palindromic subsequence:", longest\_palindromic\_subsequence(s))



6. Word break problem

def word\_break(s, wordDict):

n = len(s)

dp = [False] \* (n + 1)

dp[0] = True

wordSet = set(wordDict)

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

for j in range(i):

if dp[j] and s[j:i] in wordSet:

dp[i] = True

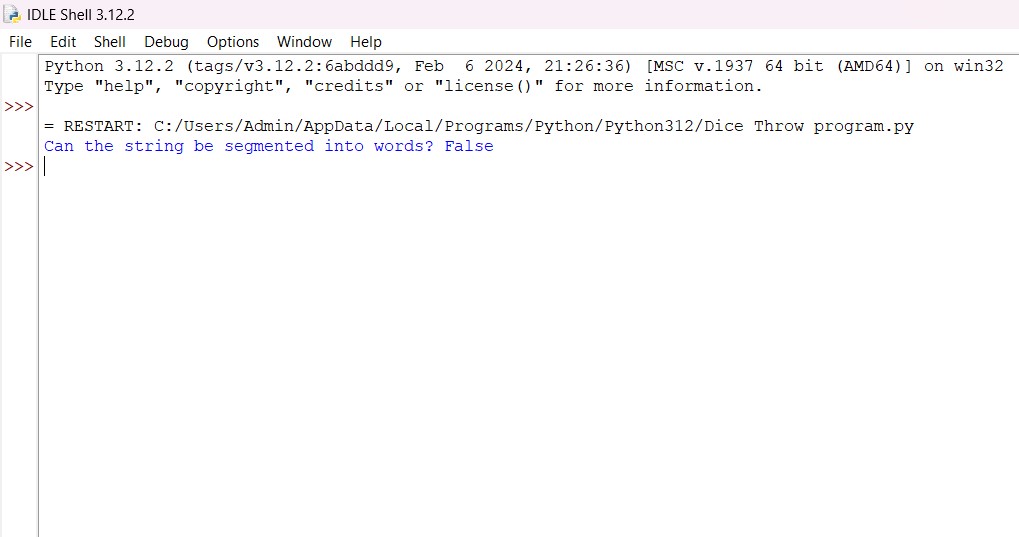
break

return dp[n]

s = "leetcode"

wordDict = ["leet", "code"]

print("Can the string be segmented into words?", word\_break(s, wordDict))



8. Word wrap problem

import sys

def word\_wrap(words, max\_width):

n = len(words)

def calculate\_cost(i, j):

total\_length = sum(len(words[k]) for k in range(i, j))

total\_length += j - i - 1

if total\_length > max\_width:

return float('inf')

return (max\_width - total\_length) \*\* 3

dp = [0] \* (n + 1)

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

for i in range(n):

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

cost[i][j] = calculate\_cost(i, j)

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

dp[i] = sys.maxsize

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

if cost[i][j] != float('inf'):

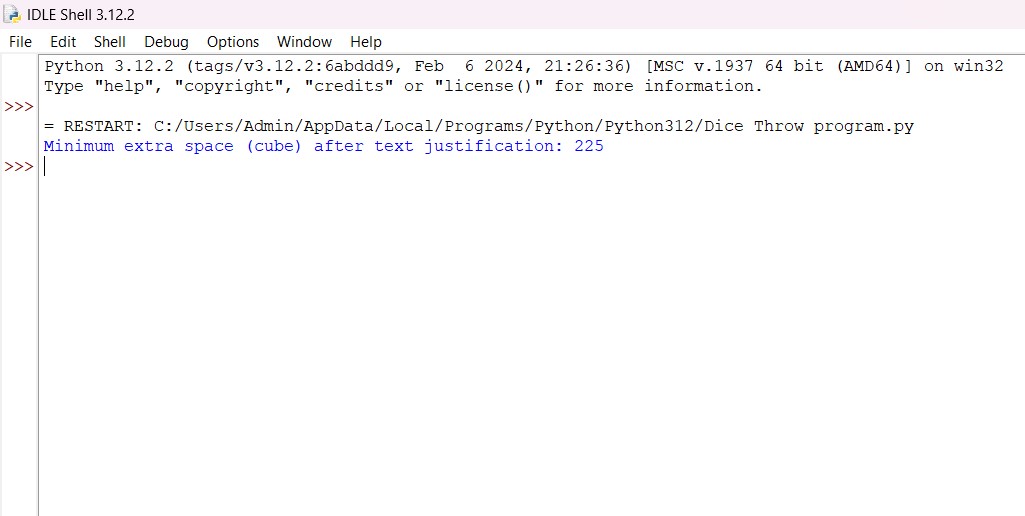
dp[i] = min(dp[i], cost[i][j] + dp[j])

return dp[0]

words = ["This", "is", "an", "example", "of", "text", "justification."]

max\_width = 16

print("Minimum extra space (cube) after text justification:", word\_wrap(words, max\_width))



9. Computing a binomial coefficient

def binomial\_coefficient(n, k):

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

for i in range(n + 1):

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

if j == 0 or j == i:

C[i][j] = 1

else:

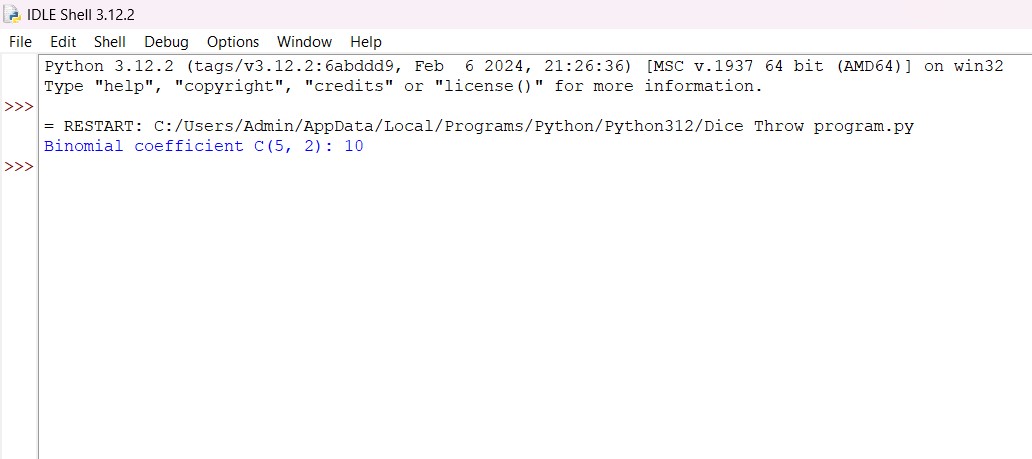
C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

return C[n][k]

n = 5

k = 2

print(f"Binomial coefficient C({n}, {k}): {binomial\_coefficient(n, k)}")



10. Knapsack problem

def knapsack(weights, values, capacity):

n = len(weights)

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

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

for j in range(capacity + 1):

if weights[i - 1] <= j:

dp[i][j] = max(dp[i - 1][j], values[i - 1] + dp[i - 1][j - weights[i - 1]])

else:

dp[i][j] = dp[i - 1][j]

return dp[n][capacity]

weights = [10, 20, 30]

values = [60, 100, 120]

capacity = 50

print("Maximum value in knapsack:", knapsack(weights, values, capacity))

