Day 3 lab python programs

1.

def max\_profit(prices):

n = len(prices)

if n <= 1:

return 0

# Initialize profit arrays for two transactions

profit1 = [0] \* n

profit2 = [0] \* n

# Calculate profit from the first transaction

min\_price = prices[0]

for i in range(1, n):

profit1[i] = max(profit1[i - 1], prices[i] - min\_price)

min\_price = min(min\_price, prices[i])

# Calculate profit from the second transaction

max\_price = prices[-1]

for i in range(n - 2, -1, -1):

profit2[i] = max(profit2[i + 1], max\_price - prices[i])

max\_price = max(max\_price, prices[i])

# Calculate the maximum profit with at most 2 transactions

max\_profit = 0

for i in range(n):

max\_profit = max(max\_profit, profit1[i] + profit2[i])

return max\_profit

# Test cases

test\_cases = [

[7, 1, 5, 3, 6, 4],

[7, 6, 4, 3, 1],

[10, 22, 5, 75, 65, 80],

[2, 30, 15, 10, 8, 25, 80],

[10, 22, 5, 75, 65, 80]

]

for prices in test\_cases:

print("Output:", max\_profit(prices))

2.

from itertools import permutations

def generate\_combinations(digits):

digit\_list = list(digits)

combinations = permutations(digit\_list)

for combo in combinations:

print(''.join(combo))

# Test cases

test\_cases = ["123", "789", "1456", "-856", "1001", "555"]

for digits in test\_cases:

print("Sample Output:")

generate\_combinations(digits)

print()

3.

def num\_good\_pairs(nums):

count = 0

freq = {}

for num in nums:

if num in freq:

count += freq[num]

freq[num] += 1

else:

freq[num] = 1

return count

# Test cases

test\_cases = [

[1, 2, 3, 1, 1, 3],

[1, 1, 1, 1],

[1, 2, 3],

[4, 5, 1, 3, 7],

[1, 2, 2, 3]

]

for nums in test\_cases:

print("Output:", num\_good\_pairs(nums))

4.

def add\_binary(a, b):

carry = 0

result = []

i, j = len(a) - 1, len(b) - 1

while i >= 0 or j >= 0 or carry:

total = carry

if i >= 0:

total += int(a[i])

i -= 1

if j >= 0:

total += int(b[j])

j -= 1

result.append(str(total % 2))

carry = total // 2

return ''.join(result[::-1])

# Test cases

test\_cases = [

("11", "1"),

("1010", "1011"),

("1111", "1010"),

("101101", "1100"),

("15", "45")

]

for a, b in test\_cases:

print("Output:", add\_binary(a, b))

5.

def min\_jumps(arr):

n = len(arr)

if n <= 1:

return 0

if arr[0] == 0:

return -1

max\_reach = arr[0]

steps = arr[0]

jumps = 1

for i in range(1, n):

if i == n - 1:

return jumps

max\_reach = max(max\_reach, i + arr[i])

steps -= 1

if steps == 0:

jumps += 1

if i >= max\_reach:

return -1

steps = max\_reach - i

return -1

# Test cases

test\_cases = [

[1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9],

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],

[2, 3, 1, 1, 4],

[1, 3, 6, 1, 0, 9],

[2, 3, 0, 1, 4]

]

for arr in test\_cases:

print("Output:", min\_jumps(arr))

6.

def is\_mirror\_number(num):

mirror\_mapping = {

'0': '0', '1': '1', '6': '9', '8': '8', '9': '6'

}

reversed\_num = ''.join([mirror\_mapping.get(digit, '') for digit in reversed(num)])

if reversed\_num == num:

return True

else:

return False

# Input

num = input("Enter the number: ")

# Remove non-numeric characters

num = ''.join(filter(str.isdigit, num))

# Check if the number can be reversed to become a mirror number

if is\_mirror\_number(num):

print("Mirror image:", num[::-1])

else:

print("No mirror image possible")

7.

def permute\_unique(nums):

def backtrack(start):

if start == len(nums):

result.append(nums[:])

return

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

if i > start and nums[i] == nums[start]:

continue

nums[start], nums[i] = nums[i], nums[start]

backtrack(start + 1)

nums[start], nums[i] = nums[i], nums[start]

nums.sort()

result = []

backtrack(0)

return result

# Test cases

test\_cases = [

[1, 1, 2],

[1, 2, 3],

[2, 8, 9, 10],

[-1, 0, 1],

[1, 1, 1]

]

for nums in test\_cases:

print("Output:", permute\_unique(nums))

8.

def group\_anagrams(strs):

anagram\_map = {}

for word in strs:

sorted\_word = ''.join(sorted(word))

if sorted\_word in anagram\_map:

anagram\_map[sorted\_word].append(word)

else:

anagram\_map[sorted\_word] = [word]

return list(anagram\_map.values())

# Test cases

test\_cases = [

["eat", "tea", "tan", "ate", "nat", "bat"],

[""],

["a"],

["banana"],

["12345"]

]

for strs in test\_cases:

print("Output:", group\_anagrams(strs))

9.

def isMatch(s, p):

dp = [[False] \* (len(p) + 1) for \_ in range(len(s) + 1)]

dp[0][0] = True

for i in range(len(s) + 1):

for j in range(1, len(p) + 1):

if p[j - 1] == '\*':

dp[i][j] = dp[i][j - 2] or (i > 0 and (s[i - 1] == p[j - 2] or p[j - 2] == '.') and dp[i - 1][j])

else:

dp[i][j] = i > 0 and (s[i - 1] == p[j - 1] or p[j - 1] == '.') and dp[i - 1][j - 1]

return dp[len(s)][len(p)]

# Test cases

test\_cases = [

("aa", "a"), # Output: False

("aa", "a\*"), # Output: True

("ab", ".\*"), # Output: True

(" aaa", "aa") # Output: True

]

for s, p in test\_cases:

print("Output:", isMatch(s, p))

10.

def min\_operations(word1, word2):

m, n = len(word1), len(word2)

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

for i in range(m + 1):

dp[i][0] = i

for j in range(n + 1):

dp[0][j] = j

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

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

if word1[i - 1] == word2[j - 1]:

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

else:

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

return dp[m][n]

# Test cases

test\_cases = [

("horse", "ros"), # Output: 3

("intention", "execution"), # Output: 5

("sunday", "saturday"), # Output: 3

("cat", "cut"), # Output: 1

("girl", "grill") # Output: 2

]

for word1, word2 in test\_cases:

print("Output:", min\_operations(word1, word2))