1. Convert the Temperature

```
def convert_temperature(celsius):
    fahrenheit = celsius * 9/5 + 32
    kelvin = celsius + 273.15
    return fahrenheit, kelvin
celsius = 25
fahrenheit, kelvin = convert_temperature(celsius)
print(fahrenheit, kelvin)
```

O/p (77.0, 298.15)

2. Number of Subarrays With LCM Equal to K

```
from math import gcd
from functools import reduce

def lcm(a, b):
    return a * b // gcd(a, b)

def number_of_subarrays_with_lcm_k(arr, k):
    def subarray_lcm(subarray):
        return reduce(lcm, subarray)

    count = 0
    for i in range(len(arr)):
        for j in range(i + 1, len(arr) + 1):
            if subarray_lcm(arr[i:j]) == k:
                count += 1
    return count
arr = [2, 3, 4, 6]
k = 12
result = number_of_subarrays_with_lcm_k(arr, k)
print(result)
```

O/p 1

3. 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 min_operations_to_sort_by_level(root):
    if not root:
        return 0

    queue = deque([root])
    operations = 0

    while queue:
        level_size = len(queue)
        current_level = []

        for _ in range(level_size):
            node = queue.popleft()
            current_level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        sorted_level = sorted(current_level)
        operations += sum(1 for i in range(len(current_level)) if current_level[i] !=
sorted_level[i])

    return operations
root = TreeNode(1, TreeNode(3, TreeNode(5), TreeNode(7)), TreeNode(2, TreeNode(6),
TreeNode(4)))
result = min_operations_to_sort_by_level(root)
print(result)

O/P 4
```

4. Maximum Number of Non-overlapping Palindrome Substrings

```
def max_non_overlapping_palindromes(s):
    n = len(s)
    dp = [[False] * n for _ in range(n)]

    for i in range(n):
        dp[i][i] = True

    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j]:
                if length == 2 or dp[i + 1][j - 1]:
                    dp[i][j] = True

    count = 0
    end = -1
    for i in range(n):
        if dp[end + 1][i]:
            count += 1
            end = i

    return count

s = "ababa"
result = max_non_overlapping_palindromes(s)
print(result)

O/P   3
```

5. **Minimum Cost to Buy Apples**

```
def min_cost_to_buy_apples(cost, quantity, k):
    n = len(cost)
    dp = [[float('inf')] * (k + 1) for _ in range(n + 1)]
    dp[0][0] = 0

    for i in range(1, n + 1):
        for j in range(k + 1):
            dp[i][j] = dp[i - 1][j]
            if j >= quantity[i - 1]:
                dp[i][j] = min(dp[i][j], dp[i - 1][j - quantity[i - 1]] + cost[i - 1])

    return dp[n][k] if dp[n][k] != float('inf') else -1

cost = [2, 3, 5]
quantity = [1, 2, 3]
k = 5
result = min_cost_to_buy_apples(cost, quantity, k)
print(result)

O/P 5
```

6. **Customers With Strictly Increasing Purchases**

7. **Number of Unequal Triplets in Array**

```
 def number_of_unequal_triplets(arr):
    n = len(arr)
    count = 0

    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if arr[i] != arr[j] and arr[j] != arr[k] and arr[i] != arr[k]:
                    count += 1
    return count
arr = [1, 2, 3, 4]
result = number_of_unequal_triplets(arr)
print(result)
```

```
O/P 4
```

8. Closest Nodes Queries in a Binary Search Tree

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def closest_nodes(root, queries):
    def inorder_traversal(node):
        return inorder_traversal(node.left) + [node.val] + inorder_traversal(node.right) if node
else []

    sorted_vals = inorder_traversal(root)
    result = []

    for q in queries:
        pos = bisect.bisect_left(sorted_vals, q)
        if pos == 0:
            result.append(sorted_vals[0])
        elif pos == len(sorted_vals):
            result.append(sorted_vals[-1])
        else:
            if abs(sorted_vals[pos] - q) < abs(sorted_vals[pos - 1] - q):
                result.append(sorted_vals[pos])
            else:
                result.append(sorted_vals[pos - 1])

    return result
root = TreeNode(4, TreeNode(2, TreeNode(1), TreeNode(3)), TreeNode(6, TreeNode(5),
TreeNode(7)))
queries = [3, 8]
result = closest_nodes(root, queries)
print(result)
```

```
O/P [3, 7]
```

9. Minimum Fuel Cost to Report to the Capital

```
def min_fuel_cost_to_capital(n, edges, price):
    from collections import defaultdict, deque
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)

    def bfs(start):
        queue = deque([start])
        visited = set([start])
        fuel_cost = 0

        while queue:
            node = queue.popleft()
            for neighbor in graph[node]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append(neighbor)
                    fuel_cost += price
        return fuel_cost

    return bfs(0)
n = 5
edges = [(0, 1), (1, 2), (1, 3), (3, 4)]
price = 2
result = min_fuel_cost_to_capital(n, edges, price)
print(result)

O/P 8
```

10. Number of Beautiful Partitions

```
def number_of_beautiful_partitions(s, k):
    n = len(s)
    dp = [[0] * (k + 1) for _ in range(n + 1)]
    dp[0][0] = 1

    for i in range(1, n + 1):
        for j in range(1, k + 1):
            for l in range(i):
                if s[l:i] == s[l:i][::-1]:
                    dp[i][j] += dp[l][j - 1]

    return dp[n][k]
s = "aab"
k = 2
result = number_of_beautiful_partitions(s, k)
print(result)                                              O/P = 2
```