**Answers:**

1. **A**
2. **A**
3. **A**
4. **B**
5. **A**
6. **A**
7. **B**
8. **A**
9. **B**
10. **D**
11. **B**
12. **B**
13. **B**
14. **B**
15. **A**
16. **B**
17. **A**
18. **B**
19. **C**
20. **B**
21. **A**
22. **B**
23. **A**
24. **A**
25. **B**
26. **A**
27. **D**

**PROGRAMS:**

def evaluate\_performance(employees):

evaluated\_employees = []

for employee in employees:

name = employee["name"]

scores = employee["scores"]

weights = employee["weights"]

# Calculate the performance score for the employee

performance\_score = sum(scores[criterion] \* weights[criterion] for criterion in scores)

evaluated\_employee = {

"name": name,

"performance\_score": performance\_score

}

evaluated\_employees.append(evaluated\_employee)

return evaluated\_employees

# Example input data

employees = [

{

"name": "John",

"scores": {

"Quality of Work": 90,

"Team Collaboration": 80,

"Punctuality": 95

},

"weights": {

"Quality of Work": 0.6,

"Team Collaboration": 0.4,

"Punctuality": 0.5

}

},

{

"name": "Alice",

"scores": {

"Quality of Work": 85,

"Team Collaboration": 90,

"Punctuality": 98

},

"weights": {

"Quality of Work": 0.6,

"Team Collaboration": 0.4,

"Punctuality": 0.5

}

}

]

# Calculate performance scores

evaluated\_employees = evaluate\_performance(employees)

# Print the result

for employee in evaluated\_employees:

print(f"{employee['name']} - Performance Score: {employee['performance\_score']}")

def calculate\_class\_average(marks):

num\_students = len(marks)

num\_subjects = len(marks[0])

class\_averages = [0] \* num\_subjects

for subject in range(num\_subjects):

subject\_total = sum(marks[i][subject] for i in range(num\_students))

class\_averages[subject] = subject\_total / num\_students

return class\_averages

def students\_report(N, M, marks):

class\_averages = calculate\_class\_average(marks)

lowest\_avg\_subject = min(enumerate(class\_averages), key=lambda x: x[1])[0]

updated\_marks = []

for student in marks:

total\_marks = sum(student)

updated\_marks.append(total\_marks - student[lowest\_avg\_subject])

return updated\_marks

# Example input data

N = 3 # Number of students

M = 4 # Number of subjects

marks = [

[80, 75, 90, 85],

[70, 65, 80, 75],

[85, 90, 95, 100]

]

# Calculate updated total marks

updated\_total\_marks = students\_report(N, M, marks)

# Print the result

for i, total\_marks in enumerate(updated\_total\_marks):

print(f"Student {i + 1}: {total\_marks} marks")

# Function to sort employees by their id in ascending order

def sort\_employees\_by\_id(employees):

return sorted(employees, key=lambda emp: emp['id'])

# Function to sort employees by their salary in descending order

def sort\_employees\_by\_salary(employees):

return sorted(employees, key=lambda emp: emp['salary'], reverse=True)

# Example list of employee records

employee\_records = [

{'id': 101, 'name': 'Aman', 'salary': 50000},

{'id': 102, 'name': 'Shubham', 'salary': 60000},

{'id': 103, 'name': 'Sagar', 'salary': 55000},

{'id': 104, 'name': 'Amit', 'salary': 62000},

]

# Sort employees by id

sorted\_by\_id = sort\_employees\_by\_id(employee\_records)

# Sort employees by salary

sorted\_by\_salary = sort\_employees\_by\_salary(employee\_records)

# Print sorted results

print("Sorted by ID:")

for employee in sorted\_by\_id:

print(f"ID: {employee['id']}, Name: {employee['name']}, Salary: {employee['salary']}")

print("\nSorted by Salary:")

for employee in sorted\_by\_salary:

print(f"ID: {employee['id']}, Name: {employee['name']}, Salary: {employee['salary']}")

def find\_kth\_prime\_product(N, prime\_numbers, K):

result = [1] # Initialize the result list with 1

while len(result) < K:

min\_product = float('inf') # Initialize the minimum product to positive infinity

for prime in prime\_numbers:

# Try multiplying the next prime with the last number in the result list

product = prime \* result[-1]

# Check if the product is greater than the last number in the result list

# and less than the current minimum product

if product > result[-1] and product < min\_product:

min\_product = product

# Add the minimum product to the result list

result.append(min\_product)

return result[-1]

# Example input

N = 3

prime\_numbers = [2, 3, 5]

K = 4

# Find the Kth number

kth\_number = find\_kth\_prime\_product(N, prime\_numbers, K)

# Print the result

print(f"The {K}th number that can be created from the prime numbers {prime\_numbers} is: {kth\_number}")

def circular\_max\_sum(children, numbers):

# Function to find the maximum sum using Kadane's algorithm

def max\_sum\_kadane(arr):

max\_ending\_here = max\_so\_far = arr[0]

for num in arr[1:]:

max\_ending\_here = max(num, max\_ending\_here + num)

max\_so\_far = max(max\_so\_far, max\_ending\_here)

return max\_so\_far

# Find the maximum sum with a circular subarray

max\_sum\_straight = max\_sum\_kadane(numbers)

# If all numbers are negative, return the maximum among them

if max\_sum\_straight < 0:

return max\_sum\_straight

# Calculate the total sum of the array

total\_sum = sum(numbers)

# Find the minimum sum with a circular subarray by inverting the numbers

# Subtract the minimum circular sum from the total sum to get the maximum circular sum

max\_sum\_circular = total\_sum - max\_sum\_kadane([-num for num in numbers])

return max(max\_sum\_straight, max\_sum\_circular)

# Example input

n = 5

numbers = [10, -4, 1, 3, 3]

# Find the continuous maximum sum

max\_sum = circular\_max\_sum(n, numbers)

# Print the result

print("The continuous maximum sum of numbers spoken is:", max\_sum)

def max\_subarray\_sum(arr, n):

max\_ending\_here = max\_so\_far = arr[0]

for i in range(1, n):

max\_ending\_here = max(arr[i], max\_ending\_here + arr[i])

max\_so\_far = max(max\_so\_far, max\_ending\_here)

return max\_so\_far

# Example input

n = 9

arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

# Find the maximum subarray sum

result = max\_subarray\_sum(arr, n)

# Print the result

print("The maximum subarray sum is:", result)

def longest\_common\_subsequence(s1, s2):

# Get the lengths of the input strings

m, n = len(s1), len(s2)

# Create a 2D table to store the length of the longest common subsequence

# dp[i][j] represents the length of LCS of s1[0...i-1] and s2[0...j-1]

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

# Fill the dp table using dynamic programming

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

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

if s1[i - 1] == s2[j - 1]:

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

else:

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

# The value in the bottom-right cell of the dp table contains the length of LCS

return dp[m][n]

# Example input

s1 = "AGGTAB"

s2 = "GXTXAYB"

# Find the length of the longest common subsequence

result = longest\_common\_subsequence(s1, s2)

# Print the result

print("The length of the longest common subsequence is:", result)

def solve\_sudoku(board):

def is\_valid\_move(row, col, num):

# Check if 'num' is already present in the same row or column

for i in range(9):

if board[row][i] == num or board[i][col] == num:

return False

# Check if 'num' is already present in the 3x3 subgrid

start\_row, start\_col = 3 \* (row // 3), 3 \* (col // 3)

for i in range(start\_row, start\_row + 3):

for j in range(start\_col, start\_col + 3):

if board[i][j] == num:

return False

return True

def solve():

for row in range(9):

for col in range(9):

if board[row][col] == 0:

for num in map(str, range(1, 10)):

if is\_valid\_move(row, col, num):

board[row][col] = num

if solve():

return True

board[row][col] = 0 # Backtrack if the solution is not found

return False

return True

solve()

# Example Sudoku board

sudoku\_board = [

[5, 3, 0, 0, 7, 0, 0, 0, 0],

[6, 0, 0, 1, 9, 5, 0, 0, 0],

[0, 9, 8, 0, 0, 0, 0, 6, 0],

[8, 0, 0, 0, 6, 0, 0, 0, 3],

[4, 0, 0, 8, 0, 3, 0, 0, 1],

[7, 0, 0, 0, 2, 0, 0, 0, 6],

[0, 6, 0, 0, 0, 0, 2, 8, 0],

[0, 0, 0, 4, 1, 9, 0, 0, 5],

[0, 0, 0, 0, 8, 0, 0, 7, 9]

]

# Solve the Sudoku

solve\_sudoku(sudoku\_board)

# Print the solved Sudoku

for row in sudoku\_board:

print(row)