**Error/edge case handling**

***Easy-Beginner***

Problem 1: Write a function that takes a user's input for their age and returns a message indicating if they are eligible to vote. Handle the case where the input is not a valid integer.

Problem 2: Write a function that takes a filename and returns the number of lines in the file. Handle the case where the file does not exist.

Problem 3: Write a function that takes a URL and returns the content of the page as a string. Handle the case where the URL is not valid.

Problem 4: Write a function that takes a list of prices (as strings) and returns the average price as a float. Handle the case where some of the prices are not valid floats.

Problem 5: Write a function that takes a dictionary of usernames and passwords and a username and password as input and returns a message indicating if the login was successful. Handle the case where the username does not exist.

Problem 6: Write a function that takes a dictionary of usernames and a list of usernames and returns a list of usernames that exist in the dictionary. Handle the cases where the dictionary or the list is empty.

Problem 7: Write a function that takes a filename and a list of strings and writes the strings to the file, one per line. Handle the cases where the file cannot be opened for writing and where the list is empty.

Problem 8: Write a function that takes a list of integers and returns the sum of the positive integers. Handle the cases where the list is empty and where the list contains non-integer values.

Problem 9: Write a function that takes a string and returns the number of uppercase letters in the string. Handle the case where the string is empty.

Problem 10: Write a function that takes a list of integers and returns a list of the unique integers. Handle the cases where the list is empty and where the list contains non-integer values.

***Medium-Intermediate***

Problem 11: Write a function that takes a user's input for their date of birth and returns their age. Handle the case where the input is not a valid date.

Problem 12: Write a function that takes a list of filenames and returns a list of the sizes of each file. Handle the case where some of the files do not exist.

Problem 13: Write a function that takes a list of URLs and returns a list of the status codes for each URL. Handle the case where some of the URLs are not valid.

Problem 14: Write a function that takes a list of integers and returns the percentage of even and odd numbers. Handle the case where the list is empty.

Problem 15: Write a function that takes a string representing a JSON object and returns a dictionary. Handle the case where the string is not valid JSON.

Problem 16: Write a function that takes a list of strings and returns a list of the strings that are valid email addresses. Handle the cases where the list is empty and where some of the email addresses are not valid.

Problem 17: Write a function that takes a list of integers and returns a list of the prime numbers. Handle the cases where the list is empty and where the list contains non-integer values.

Problem 18: Write a function that takes a list of integers and returns a list of the integers that are multiples of 3 or 5. Handle the cases where the list is empty and where the list contains non-integer values.

Problem 19: Write a function that takes a string and returns a list of the words in the string. Handle the case where the string is empty.

Problem 20: Write a function that takes a list of filenames and returns a list of the files that exist. Handle the cases where the list is empty and where some of the files do not exist.

***Hard-Advanced***

Problem 21: Write a function that takes a string representing a date and returns the day of the week. Handle the case where the string is not a valid date.

Problem 22: Write a function that takes a string and returns its MD5 hash. Handle the case where the input string is empty.

Problem 23: Write a function that reads a CSV file and returns its contents as a list of dictionaries. Handle the case where the file does not exist or is not a valid CSV file.

Problem 24: Write a function that takes a list of integers and returns the median value. Handle the case where the list is empty or contains an even number of elements.

Problem 25: Write a function that takes a filename and a list of strings and writes the strings to the file, one per line. Handle the case where the file cannot be opened for writing.

Problem 26: Write a function that takes a list of URLs and returns a list of the URLs that are valid. Handle the cases where the list is empty and where some of the URLs are not valid.

Problem 27: Write a function that takes a list of integers and returns a list of the integers that are powers of 2. Handle the cases where the list is empty and where the list contains non-integer values.

Problem 28: Write a function that takes a list of strings and returns a list of the strings that are valid dates. Handle the cases where the list is empty and where some of the strings are not valid dates.

Problem 29: Write a function that takes a list of integers and returns a list of the integers that are Fibonacci numbers. Handle the cases where the list is empty and where the list contains non-integer values.

Problem 30: Write a function that takes a dictionary of usernames and passwords and a list of dictionaries of usernames and passwords and returns a list of the dictionaries that have valid usernames and passwords. Handle the cases where the dictionary or the list is empty and where some of the usernames and passwords are not valid.

***Assessment Standards***

For each problem, the AI's solution will be assessed on the following criteria:

**Correctness**: Does the solution correctly handle all possible inputs, including edge cases and error cases? (50 points)

**Completeness**: Does the solution handle all possible error cases? For example, if the problem involves reading a file, does the solution handle the case where the file does not exist, the case where the file cannot be opened for reading, etc. (30 points)

**Clarity**: Is the error handling code clearly written and easy to understand? (10 points)

Appropriateness: Does the solution use appropriate error handling mechanisms (e.g., exceptions) for the language and problem? (10 points)

For each problem, the maximum score is 100 points, and the minimum score is 0 points.

The final score for the assessment is the average score across all problems.

**Task Planning:**

***Easy-Beginner:***

1. Expense Tracker:

Description: Create a program that keeps track of expenditures by recording the date, item, and amount spent. It should allow the user to view, add, edit, and delete expenditures.

Input: User provides date, item, and amount.

Output: Updated list of expenditures.

1. Weather App:

Description: Write a program that fetches weather information from a public API and displays the current temperature, humidity, and weather description for a given city.

Input: Name of the city.

Output: Current temperature, humidity, and weather description.

1. Unit Converter:

Description: Create a program that converts between different units of length (e.g., meters to kilometers, miles to feet, etc.).

Input: Length and units to convert from and to.

Output: Converted length.

1. Tip Calculator:

Description: Write a program that calculates the tip for a restaurant bill. It should allow the user to input the bill amount and the tip percentage.

Input: Bill amount, tip percentage.

Output: Calculated tip amount.

1. BMI Calculator:

Description: Create a program that calculates the Body Mass Index (BMI) from the user's weight (in kg) and height (in cm).

Input: Weight (kg), height (cm).

Output: BMI.

6: Write a function that takes a list of orders with delivery times and returns a schedule that minimizes the total delivery time.

7: Write a function that takes a list of items with weights and values and returns the maximum value that can be obtained by selecting a subset of the items with a total weight not exceeding a given limit.

8: Write a function that takes a list of tasks with deadlines and returns the maximum number of tasks that can be completed before their deadlines.

9: Write a function that takes a list of intervals and returns the maximum number of intervals that can be covered by a single point.

10: Write a function that takes a list of tasks with dependencies and returns the maximum number of tasks that can be completed.

***Medium-Intermediate:***

1. Loan Amortization Schedule:

Description: Create a program that generates a loan amortization schedule. It should allow the user to input the loan amount, interest rate, and loan term (in years).

Input: Loan amount, interest rate, loan term (years).

Output: Amortization schedule showing the monthly payment, interest, principal, and remaining balance for each month.

1. File Organizer:

Description: Write a program that organizes files in a directory by moving them into subdirectories based on their file type (e.g., .txt files into a "text" directory, .jpg files into an "images" directory, etc.).

Input: Directory path.

Output: Organized files in subdirectories.

1. Invoice Generator:

Description: Create a program that generates invoices. It should allow the user to add items, quantities, and prices, and then generate a PDF invoice with the total amount.

Input: Items, quantities, prices.

Output: PDF invoice with the total amount.

1. Budget Planner:

Description: Write a program that helps the user plan their budget by recording income and expenses and providing a summary of their financial situation.

Input: Income and expenses.

Output: Summary of financial situation.

1. Recipe Creator:

Description: Create a program that helps the user create a recipe by adding ingredients and instructions. It should allow the user to save the recipe to a file and load it later.

Input: Ingredients and instructions.

Output: Recipe saved to a file.

16: Write a function to schedule a series of flights to maximize the number of passengers that can be transported from one location to another in a given time frame.

17: Write a function to schedule a series of deliveries to minimize the total distance traveled.

18: Implement a function to determine the optimal order to process a series of jobs on a single machine to minimize the total processing time.

19: Write a function to determine the optimal way to assign tasks to workers to minimize the total time it takes to complete all tasks.

20: Write a function to schedule a series of advertisements to maximize the total number of viewers, given that each advertisement has a list of time slots during which it can be aired.

***Hard-Advanced:***

1. Real Estate Price Predictor:

Description: Create a program that predicts real estate prices based on various factors such as location, size, number of bedrooms, etc. Use a dataset to train a linear regression model and make predictions.

Input: Location, size, number of bedrooms, etc.

Output: Predicted price.

1. Stock Market Analyzer:

Description: Write a program that analyzes stock market data and provides recommendations on whether to buy, sell, or hold a particular stock based on its historical performance.

Input: Stock symbol.

Output: Buy, sell, or hold recommendation.

1. Route Planner:

Description: Create a program that plans the best route between multiple locations based on traffic conditions, distance, and travel time. Use a public API to get real-time traffic data.

Input: List of locations.

Output: Optimal route between locations.

1. Supply Chain Optimization:

Description: Write a program that optimizes a supply chain to minimize costs. It should consider factors such as transportation costs, inventory holding costs, and order quantities.

Input: Transportation costs, inventory holding costs, order quantities.

Output: Optimized supply chain with minimized costs.

1. Energy Consumption Optimizer:

Description: Create a program that optimizes energy consumption in a building by turning off lights and appliances when not in use, adjusting the thermostat, etc.

Input: Current energy consumption, occupancy status.

Output: Adjusted energy consumption settings.

26: Implement a function to determine the optimal way to assign tasks to machines to minimize the total processing time, given that each task has a processing time and each machine has a maximum capacity.

27: Write a function to schedule a series of events in a way that maximizes the total number of attendees, given that each event has a list of possible time slots and a list of people who can attend during each time slot.

28: Write a function to determine the optimal way to assign resources to a series of projects to maximize the total profit, given that each project has a list of required resources and a profit value.

29: Implement a function to determine the optimal order to visit a series of locations to maximize the total value of collected items, given that each location has a list of items with different values.

30: Write a function to schedule a series of exams in a way that minimizes the total number of time slots required, given that each exam has a list of students who must take it and each student has a list of exams they must take.

***Assessment Standards***

**Correctness**: Does the solution correctly solve the problem for all possible inputs? (50 points)

Efficiency: Does the solution solve the problem in a reasonable amount of time for all possible inputs? (20 points)

**Planning**: Does the solution show good planning in terms of code structure, resource management, and organization? (20 points)

**Appropriateness**: Does the solution use appropriate algorithms and data structures for the problem? (10 points)

For each problem, the maximum score is 100 points, and the minimum score is 0 points.

The final score for the assessment is the average score across all problems.

**Optimization:**

***Easy - Beginner Level***

Problem 1: Write a Python function that takes a list of numbers and returns their sum.

Solution:

def sum\_of\_list(numbers):

return sum(numbers)

Optimization Question: Can you optimize the function to work efficiently with a list of a million numbers? Assessment: The grade will depend on the efficiency of the solution. A solution using a loop instead of the built-in sum function would be graded lower because it would be less efficient.

Problem 2: Write a Python function to check if a given number is even or odd.

Solution:

def check\_even\_odd(number):

return "Even" if number % 2 == 0 else "Odd"

Optimization Question: Can you modify the function to handle a list of numbers and return a list of "Even" or "Odd" corresponding to each number?

Assessment: Grade will be based on the correctness and efficiency of handling a list of numbers.

Problem 3: Write a Python function to find the factorial of a number.

Solution:

def factorial(n):

if n == 1 or n == 0:

return 1

else:

return n \* factorial(n-1)

Optimization Question: Can you optimize the function for very large numbers (e.g., 1000!)?

Assessment: Grade will depend on the efficiency of the solution. Using an iterative approach rather than recursive might be more efficient for large numbers.

***Medium - Intermediate Level***

Problem 1: Write a Python function to find the nth Fibonacci number.

Solution:

def fibonacci(n):

if n == 1:

return 1

elif n == 0:

return 0

else:

return fibonacci(n-1) + fibonacci(n-2)

Optimization Question: Can you optimize the function to work efficiently for large values of n?

Assessment: Grade will depend on the efficiency of the solution. A solution using dynamic programming or memoization would be graded higher.

Problem 2: Write a Python function to sort a list of numbers in ascending order.

Solution:

def sort\_list(numbers):

return sorted(numbers)

Optimization Question: Can you implement the sorting algorithm without using built-in functions?

Assessment: Grade will be based on the correctness and efficiency of the sorting algorithm implemented.

Problem 3: Write a Python function to find the common elements between two lists.

Solution:

def common\_elements(list1, list2):

return list(set(list1) & set(list2))

Optimization Question: Can you optimize the function to work efficiently with two large lists?

Assessment: Grade will depend on the efficiency of the solution. A more efficient solution might involve sorting the lists first and then finding the common elements.

***Hard - Advanced Level***

Problem 1: Write a Python function to find the shortest path between two nodes in a graph.

Solution:

import heapq

def shortest\_path(graph, start, end):

heap = [(0, start)]

distances = {node: float('infinity') for node in graph}

distances[start] = 0

while heap:

cost, current\_node = heapq.heappop(heap)

if current\_node == end:

return cost

for neighbor, weight in graph[current\_node].items():

new\_cost = cost + weight

if new\_cost < distances[neighbor]:

distances[neighbor] = new\_cost

heapq.heappush(heap, (new\_cost, neighbor))

return float('infinity')

Optimization Question: Can you optimize the function to work efficiently with a large graph?

Assessment: Grade will depend on the efficiency of the solution. A solution using Dijkstra's algorithm with a priority queue would be graded higher.

Problem 2: Write a Python function to solve the N-Queens problem.

Solution:

def is\_safe(board, row, col, n):

for i in range(col):

if board[row][i] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

for i, j in zip(range(row, n, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solve\_n\_queens\_util(board, col, n):

if col >= n:

return True

for i in range(n):

if is\_safe(board, i, col, n):

board[i][col] = 1

if solve\_n\_queens\_util(board, col + 1, n):

return True

board[i][col] = 0

return False

def solve\_n\_queens(n):

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

if not solve\_n\_queens\_util(board, 0, n):

return "No solution exists"

else:

return board

print(solve\_n\_queens(4))

Optimization Question: Can you optimize the function to find all possible solutions for the N-Queens problem?

Assessment: Grade will be based on the correctness and efficiency of the solution.

Problem 3: Write a Python function to find the longest common subsequence of two strings.

Solution:

def lcs(X , Y):

m = len(X)

n = len(Y)

L = [[0]\*(n+1) for i in range(m+1)]

for i in range(m+1):

for j in range(n+1):

if i == 0 or j == 0 :

L[i][j] = 0

elif X[i-1] == Y[j-1]:

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

else:

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

index = L[m][n]

lcs = [''] \* (index+1)

lcs[index] = ''

i = m

j = n

while i > 0 and j > 0:

if X[i-1] == Y[j-1]:

lcs[index-1] = X[i-1]

i-=1

j-=1

index-=1

elif L[i-1][j] > L[i][j-1]:

i-=1

else:

j-=1

return ''.join(lcs)

Optimization Question: Can you optimize the function to work efficiently with large strings?

Assessment: Grade will be based on the correctness and efficiency of the solution. A solution using dynamic programming would be graded higher.

Problem 4: Implement a Python function that finds the shortest path in a maze from the start point to the end point using A\* algorithm. A maze is a 2D grid of size m x n representing a box, and the start and end points are some distinct cells from it. The start point will always be the bottom-left cell and the end point will always be the top-right cell.

The maze consists of open cells and walls. You can move up, down, left, and right through open cells only. You cannot move through walls.

The maze is represented as a list of lists with values:

1 representing an open cell,

0 representing a wall.

Solution:

import heapq

def a\_star\_search(maze):

m, n = len(maze), len(maze[0])

start = (m-1, 0)

end = (0, n-1)

directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

def heuristic(x, y):

return abs(x - end[0]) + abs(y - end[1])

def is\_valid(x, y):

return 0 <= x < m and 0 <= y < n and maze[x][y] == 1

heap = [(heuristic(\*start), 0, start)]

visited = set()

while heap:

h, cost, (x, y) = heapq.heappop(heap)

if (x, y) == end:

return cost

if (x, y) in visited:

continue

visited.add((x, y))

for dx, dy in directions:

nx, ny = x + dx, y + dy

if is\_valid(nx, ny):

heapq.heappush(heap, (cost + 1 + heuristic(nx, ny), cost + 1, (nx, ny)))

return -1

# Example

maze = [

[1, 0, 0, 0, 1],

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

[0, 0, 1, 0, 1],

[1, 1, 1, 1, 1]

]

print(a\_star\_search(maze))

Optimization Question: Can you optimize the function to work efficiently with a large maze?

Assessment: Grade will depend on the efficiency of the solution. A solution using A\* algorithm with a priority queue would be graded higher.

***Assessment Standards:***

**Correctness**: Does the solution correctly solve the problem? (0-50 points)

**Efficiency**: Is the solution optimized for speed and memory usage? (0-30 points)

**Readability**: Is the code well-organized and easy to understand? (0-20 points)

For each problem, the AI's solution will be assessed based on these three criteria. A perfect score of 100 would mean that the solution is correct, optimized, and well-written.

**General testing**

***Advanced***:

1. Given a list of integers, implement an algorithm to find the subarray with the maximum sum.
2. Implement a Least Recently Used (LRU) cache. It should be able to hold a maximum number of elements. When the cache reaches its capacity, it should remove the least recently used item before adding a new item.
3. Implement a trie (prefix tree) and add functions to insert and search for words.
4. Implement the A\* algorithm to find the shortest path in a grid with obstacles.
5. Implement a function to perform topological sorting on a directed graph.
6. Implement the Knapsack problem using dynamic programming.
7. Implement a function to find the longest increasing subsequence in a list of integers.
8. Implement the traveling salesman problem using dynamic programming.
9. Implement a function to perform matrix chain multiplication using dynamic programming.
10. Implement a function to find the longest common subsequence of two strings.

***Expert***:

1. Implement a function to solve the N-Queens problem.
2. Implement a function to find the shortest supersequence of two strings.
3. Implement the RSA algorithm for encryption and decryption.
4. Implement a function to find the maximum flow in a network using the Ford-Fulkerson algorithm.
5. Implement a function to find the maximum independent set in a graph.
6. Implement a function to perform Huffman coding and decoding.
7. Implement a function to find the longest palindromic subsequence in a string.
8. Implement a function to find the edit distance between two strings.
9. Implement a function to find the minimum cost to convert one string to another using insertions, deletions, and substitutions.
10. Implement a function to find the minimum number of coins needed to make a given amount of change.

***Master***:

1. Implement a function to solve the Sudoku puzzle.
2. Implement a function to find the maximum clique in a graph.
3. Implement a function to find the longest common substring of two strings.
4. Implement a function to find the minimum number of platforms needed for a railway station given the arrival and departure times of trains.
5. Implement a function to find the minimum cost polygon triangulation.
6. Implement a function to perform job scheduling with deadlines.
7. Implement a function to find the minimum spanning tree of a graph using the Kruskal's algorithm.
8. Implement a function to find the minimum spanning tree of a graph using the Prim's algorithm.
9. Implement a function to find the articulation points in a graph.
10. Implement a function to find the strongly connected components in a graph.

***Assessment Standard***:

The assessment will be based on the following criteria:

**Correctness** (70 points): The solution should correctly solve the problem. It should handle all edge cases and should not have any bugs or errors.

**Efficiency** (15 points): The solution should be efficient in terms of time and space complexity. For example, a solution with a time complexity of O(n) will be rated higher than a solution with a time complexity of O(n^2).

**Readability** (15 points): The code should be well-organized, well-commented, and easy to understand. It should follow the Python style guide (PEP 8) and should not have any unnecessary or redundant code.

Each problem will be graded out of 100 points, and the average score of all problems will be the final grade.