Task 1: - Usage of hash function in python data structure

Hash functions have a vital role in various Python data structures and algorithms. They provide a way to convert complex data into fixed-size values (hash codes) that can be used for efficient storage, retrieval, and comparison. Here's how hash functions are used in different Python data structures:

1. Dictionaries (dict): Hashing is the foundation of dictionaries in Python. When you use a key to access or store a value in a dictionary, a hash function is applied to the key. The resulting hash code is used to determine the index where the value should be stored or retrieved. This enables constant-time average-case operations for key-based operations.

2. Sets: Hashing is also integral to sets in Python. Sets store unique elements, and hash functions help organize these elements for efficient membership checks, insertions, and deletions.

3. Custom Hashable Objects: When you create custom classes, you can define your own hash function (`__hash__` method) and equality comparison (`__eq__` method). This allows instances of your class to be used as keys in dictionaries or elements in sets.

4. Caching and Memoization: Hashing functions are used in caching and memoization to store previously computed results based on their input parameters. By associating hash codes with computed values, you can quickly retrieve cached results and avoid redundant calculations.

5. Hash-Based Algorithms: Hash functions are employed in various algorithms like hash-based searching, password hashing, data integrity verification, and more.

Hash functions enable efficient data storage and retrieval by distributing data uniformly across available memory space. They help minimize collisions and provide a mechanism for quickly identifying and accessing stored data.

Task 2: - Graph implementation in many ways

1. Adjacency Matrix:

In this representation, a graph is represented as a 2D matrix. The value at `matrix[i][j]` indicates whether there is an edge between vertex `i` and vertex `j`.

2. Adjacency List:

In this representation, a graph is represented as a dictionary where each vertex is a key, and its corresponding value is a list of vertices to which it is connected.

3. Edge List:

In this representation, a graph is stored as a list of edges, where each edge is a tuple `(v1, v2)` indicating a connection between vertices `v1` and `v2`.

4. Object-Oriented Approach:

You can represent vertices and edges as objects, creating a more structured graph implementation.