1-Hash functions are used in Python data structures to store and retrieve data quickly and efficiently. They work by converting a data item (such as a string, number, or object) into a unique hash value. This hash value is then used to index the data item into a hash table.

Hash tables are a type of data structure that allows for fast lookups of data items. This is because the hash function ensures that similar data items are stored in the same bucket (or slot) in the hash table. This means that when you search for a data item, you only need to check a small number of buckets instead of searching the entire hash table.

Here are some of the ways that hash functions are used in Python data structures:

* Dictionaries: Dictionaries are a key-value data structure that use hash functions to store and retrieve data items. The key is used to generate a hash value, which is then used to index the data item into the dictionary.
* Sets: Sets are a collection of unique elements that use hash functions to store and retrieve data items. The hash function ensures that each data item in the set has a unique hash value, which prevents duplicate elements from being stored in the set.
* Hash tables: Hash tables are a general-purpose data structure that can be used to store and retrieve any type of data. They use hash functions to convert data items into hash values, which are then used to index the data items into the hash table.

In addition to these data structures, hash functions are also used in other Python applications, such as:

* File hashes: Hash functions can be used to generate checksums of files. This can be used to verify the integrity of a file after it has been transferred or stored.
* Password hashing: Hash functions can be used to store passwords in a secure manner. This is because it is very difficult to reverse a hash function, so even if a hacker gains access to the password hashes, they will not be able to learn the original passwords.
* Digital signatures: Hash functions can be used to create digital signatures. This is a way to verify the authenticity of a message or document. The message or document is hashed, and then the hash value is signed with a private key. The signature can then be verified with the public key to ensure that the message or document has not been tampered with.

Hash functions are a powerful tool that can be used to improve the performance and security of Python data structures and applications.

* 2-Adjacency matrix: An adjacency matrix is a square matrix that represents the connections between nodes in a graph. The rows and columns of the matrix correspond to the nodes in the graph, and the value at any given row and column indicates whether there is an edge between the two nodes.
* Adjacency list: An adjacency list is a list of lists that represent the connections between nodes in a graph. Each list in the adjacency list corresponds to a node in the graph, and the elements of the list are the nodes that are adjacent to the current node.
* Edge list: An edge list is a list of tuples that represent the edges in a graph. Each tuple in the edge list consists of two elements: the two nodes that are connected by the edge.

def create graph(vertices):

adjacency matrix = [[0 for \_ in range(vertices)] for \_ in range(vertices)]

return adjacency matrix.

def add edge (adjacency matrix, u, v):

adjacency matrix[u][v] = 1

adjacency matrix[v][u] = 1

def print graph (adjacency matrix):

for i in range (Len (adjacency matrix)):

for j in range (Len (adjacency matrix [0])):

print (adjacency matrix[i][j], end=" ")

print ()

if \_\_name\_\_ == "\_\_main\_\_":

vertices = 5

adjacency matrix = create graph(vertices)

add edge (adjacency matrix, 0, 1)

add edge (adjacency matrix, 0, 2)

add edge (adjacency matrix, 1, 2)

add edge (adjacency matrix, 2, 3)

add edge (adjacency matrix, 3, 4)

print graph (adjacency matrix)