1.What is a Hash table or a Hashmap in Python?

In computer science, a Hash table or a HashMap is a type of data structure that maps keys to its value pairs (implement abstract array data types). It basically makes use of a function that computes an index value that in turn holds the elements to be searched, inserted, removed, etc. This makes it easy and fast to access data. In general, hash tables store key-value pairs and the key is generated using a hash function.

Hash tables or has maps in Python are implemented through the built-in dictionary data type. The keys of a dictionary in Python are generated by a hashing function. The elements of a dictionary are not ordered and they can be changed.

An example of a dictionary can be a mapping of employee names and their employee IDs or the names of students along with their student IDs.

Moving ahead, let's see the difference between the hash table and hashmap in Python.

Creating Dictionaries:
Dictionaries can be created in two ways:

- Using curly braces ({})
- Using the *dict()* function

Using curly braces:
Dictionaries in Python can be created using curly braces as follows:

EXAMPLE:

```
my_dict={'Dave' : '001' , 'Ava':
'002' , 'Joe': '003'}
print(my_dict)
type(my_dict)
```

OUTPUT:

```
{'Dave': '001', 'Ava': '002', 'Joe': '003'}
dict
```

Performing Operations on Hash tables using Dictionaries:

There are a number of operations that can be performed on has tables in

Python through dictionaries such as:

- Accessing Values
- Updating Values
- Deleting Element

Accessing Values:

The values of a dictionary can be accessed in many ways such as:

- Using key values
- Using functions
- Implementing the for loop

Using key values:
Dictionary values can be accessed using the key values as follows:

EXAMPLE:

```
my_dict={'Dave' : '001' , 'Ava': '002' , 'Joe': '003'}
my_dict['Dave']
```

OUTPUT: '001'

Using functions:

There are a number of built-in functions that can be used such as get (), keys (), values (), etc.

2. How to implement a graph in Python

A **graph** is a data structure that consists of vertices that are connected via edges. It can be implemented with an:

1. Adjacency list

For every vertex, its adjacent vertices are stored. In the case of a weighted graph, the edge weights are stored along with the vertices.

2. Adjacency matrix

The row and column indices represent the vertices: ������[�][�]=1*matrix*[*i*][*j*]=1 means that there is an edge from vertices �*i* to �*j*, and ������[�][�]=0*matrix*[*i*][*j*]=0 denotes that there is no edge between �*i* and �*j*. For a weighted graph,the edge weight is usually written in place of

Implementation

1. Using an adjacency list

The following code implements a graph using an adjacency list: `add_vertex(v)` adds new vertex `v` to the graph, and `add_edge(v1, v2, e)` adds an edge with weight `e` between vertices `v1v2`

`2. Using an adjacency matrix`

The following code implements a graph using an adjacency matrix: `add_vertex(v)` adds new vertex `v` to the graph, and `add_edge(v1, v2, e)` adds an edge with weight `e` between vertices `v1` and `v2`.