Hash map in python

A hashmap is a data structure that maps keys to their respective value pairs. It makes it easy to find values that are associated with their keys. There is no difference between a dictionary and a hashmap. In fact, to use hashmap, you need to understand Python dictionaries

What is a hashmap in Python?

Hashmaps are data structures that are used to store data in a key-value format. They are commonly used because they provide fast lookup times, which can be important in applications where you need to quickly retrieve data. Additionally, they can be used to implement other data structures, such as sets and caches.

Hashmaps are very useful to software engineers. They allow for reading and working with large datasets when looking for keys or values and make it faster to search for specific values in 0(1).

Here's a Python hashmap example to better understand its use:

Let's say you need to quickly find an element in a large dataset, such as a school database. You have to efficiently keep track of the scores of various students. Each student has a matriculation number, which is the key, and a Grade Point Average (GPA), which is the value. Using a student's ID, you can check their GPA. You can calculate their performance, compare grades, and see their progress.

What are hash functions in a hashmap?

A hash function is a function that takes "key" as input and returns a "hash value" or "hash" as a fixed-size output. The main importance of a hash function is that you can give the same input and the function will always produce the same output. This is important because it allows the hashmap to quickly determine where a particular key-value pair should be stored in the map.

The major reason hash functions are mostly in hashmaps is because they allow the latter to efficiently determine the location of a particular key-value pair in the map. This is done by first applying the hash function to the key to return a hash value, and then using the hash value to calculate the index at which the key-value pair should be stored in the map.

How the hash function works

The function takes a string as input and returns an integer hash value for that string. It does this by looping through each character in the key, multiplying the hash value by a prime number, and adding the ASCII value of the character to the hash value.

```
def my_hash_func(key: str) -> int:
   hash_value = 17
   for ch in key:
hash_value = hash_value * 31 + ord(ch)
   return hash_value
```

This is a very basic hash function and is not intended for use in production. It's just an example to illustrate how a hash function might work in Python. For a more efficient hash function, you should use one of the built-in options provided by Python, such as the hash() function.

How to add data to a hashmap

You can use the update() method to add data to a hashmap in Python. It takes a dictionary as an argument and adds all the key-value pairs from it to the hashmap. For example:

```
student_list = {}
students = {
   "Grace": "001",
   "Hanna": "002",
   "John": "003",
   "Mike": "004",
   "James": "005"}
student_list.update(students)
print(student_list)
# {'Grace': '001', 'Hanna': '002', 'John': '003', 'Mike': '004', 'James':
'005'}
```

Alternatively, you can add a single key-value pair to the hashmap using the syntax hashmap[key] = value.

```
student_data = {}
student_data["Grace"] = "001"
student_data["Hanna"] = "002"
student_data["John"] = "003"
student_data["Mike"] = "004"
student_data["James"] = "005"
print(student_data)
# {'Grace': '001', 'Hanna': '002', 'John': '003', 'Mike': '004', 'James':
'005'}
```

Deleting elements in a hashmap

In Python, you can use the del keyword to delete an element of a key-value pair from a dict object, which is the equivalent of a HashMap in Python. Here is an example:

```
# Remove a key-value pair from the dictionary
del students['003']# Print the updated dictionary
print(students)  # Output: {'001': 'John', '002': 'Mike', '004': 'Hanna',
'005': 'Grace'}
```

You can also use the pop() method of a dictionary object, which is also the equivalent of a HashMap in Python, to remove a key-value pair from the dictionary. The pop() method takes the key of the item to be removed as its argument and returns the value associated with that key. For example:

```
Define a dictionary
students = {"001": "John", "002": "Mike", "003": "Joy", "004": "Hanna",
"005": "Grace"}
# Remove a key-value pair from the dictionary
value = students.pop('001')
# Print the removed value
print(value)  # Output: John
# Print the updated dictionary
print(students)  # Output: {'002': 'Mike', '003': 'Joy', '004': 'Hanna',
'005': 'Grace'}
```

Note that if you try to remove a key-value pair using a key that does not exist in the dictionary, you will get a KeyError exception. To avoid this, you can use the pop() method with a default value.

```
students = {"001": "John", "002": "Mike", "003": "Joy", "004": "Hanna",
"005": "Grace"}


# Use the pop() method with a default value
value = students.pop('009', 'default_value')


# Print the value
print(value)  # Output: default_value
```