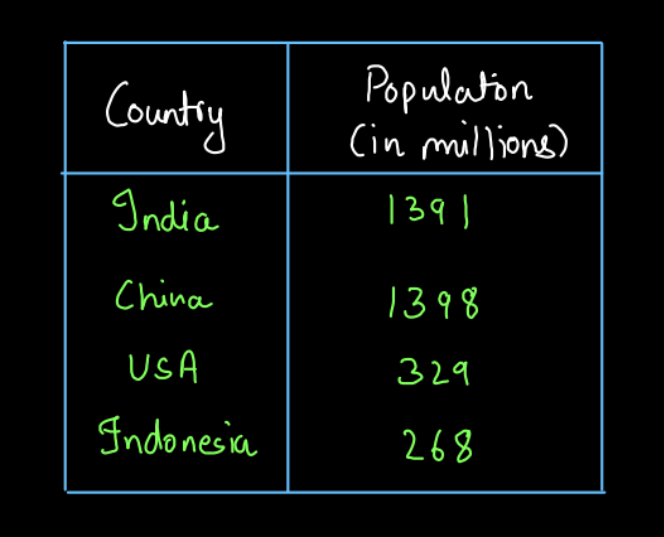
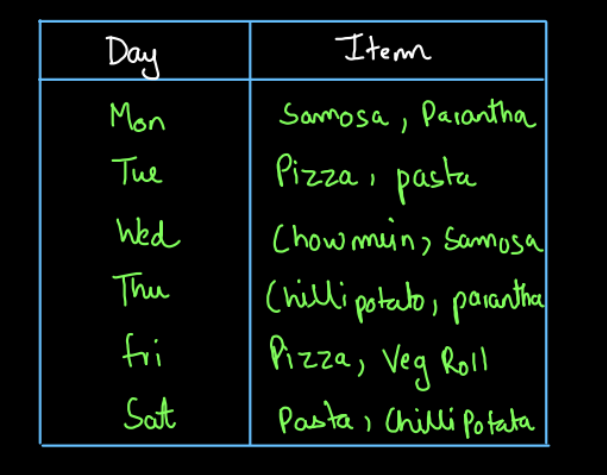
**1. What is a Hashmap?**

A hashmap is a data structure which has an amazing property that most of the operations which we perform on it are done in O(1) time complexity. The data is stored in a hash-map in the form of **key-value pairs**. For instance: consider the following data about the population of various countries:



If the data stored above in the table (fig-1) is stored in a hashmap then it is stored in the form of a key-value pair. We can decide what can be the key and what can be the value. Here, it seems pretty obvious that the name of a country should be the key and its population should be the value. Why? Think about this!!!

Another example of the key-value pairs (which you might be interested in) is for a college canteen. Let's say there are two tables. One is for which day, what is the menu and other is say for the price of each item:





These two tables can be stored in two hashmaps. One will have both key and value as a string (menu table) and the other will have key as a string and value as an integer. So, we can have any combination of data types when we talk about hashmaps. Also, there is one important thing to note in the price of the items table above (fig-3). Three items, vegetarian-roll, pizza and pasta, all have a price of 45. So, different keys can have the same values. But can we have multiple pasta keys or multiple pizza keys in the hashmap? You will get to know this very soon. Let's discuss the creation of a hashmap and perform some operations on it.

**2. Create a Hashmap**

Let us try to write the code to create a hashmap. Like any other data structure we will create it with the help of the **new** keyword. The code to create a hashmap is given below:

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args)

{

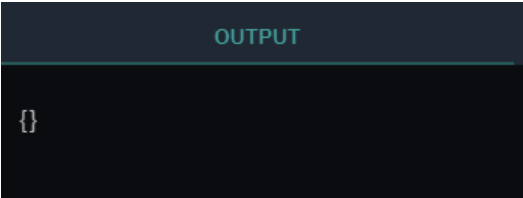
HashMap< String, Integer> hm = new HashMap< >();

System.out.println(hm);

}

}

We have created a hashmap but we have not inserted any values in it. So, we should get an empty hashmap when we print it. So, the output that we get is:



This is what an empty hashmap looks like. So, let's now try to insert some values into it and then display it.

**3. Put into Hashmap**

**Time Complexity:**

**O(1)**

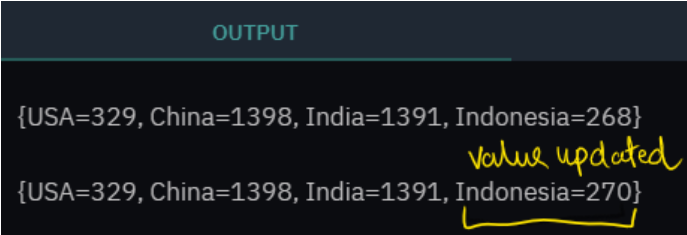
We can use the put(key,value) function to put some values into the hashmap. There are two possible cases while we are using the put(key,value) function in a hashmap.

1. **The key is not present:** If the key is not present in the hashmap it will get inserted.
2. **The key is already present:** If the key is already present, we cannot insert the same key again. The value of the existing key will get updated in the hashmap.

Let us insert the population of various countries into the hashmap:



We first try to insert the four countries with their population and print the hashmap. These will get inserted as there is no key already present with the same name. Now, we try to again insert the key "Indonesia" which is already present. So, its value will get updated. (Have a look at the output given below)



Did you notice something strange in the output? The hashmap has changed the order in which we inserted the data into it. This is something that we cannot control in a hashmap. We cannot control the order of occurrence of elements in a hashmap. Now that we have seen the put(key,value) method, let's see the get method of the hashmaps.

**4. Get a value in Hashmap**

**Time Complexity:**

**O(1)**

We can use the get(key) function in a hashmap to get the value corresponding to a particular key in the hashmap. There can be two possible cases for this:

1. **If the key exists:** If the key exists, you will get the value of that key by using this function.
2. **If the Key doesn't Exist:** If the key doesn't exist, this function will return null.

Let us try to get from the hashmap that we created above, both, a value for which key exists and one for which it doesn't.

**Java Code (Get function)**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args)

{

HashMap< String, Integer> hm = new HashMap< >();

hm.put("India", 1391);

hm.put("China", 1398);

hm.put("USA", 329);

hm.put("Indonesia", 268);

//Printing the get(key) when key exists

System.out.println("Value for India " + hm.get("India"));

//Printing the get(key) when kkey doesn't exist

System.out.println("Value for Utopia " + hm.get("Utopia"));

}

}

When we try to get the value for key "India", it already exists and we get the value that we set earlier but key "Utopia" doesn't exist in the hashmap so we get null for it (Refer to the output given below)



Now, let us see another function of hashmaps called containsKey.

**5. Contains Key in Hashmap**

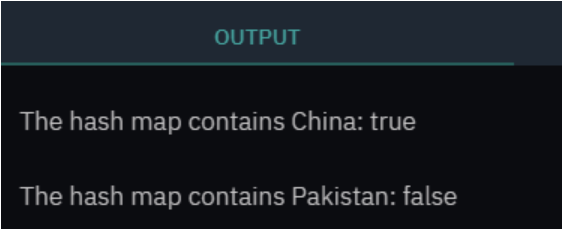
**Time Complexity:**

**O(1)**

This is a Boolean function. We pass a particular key value as a parameter to this function. If the hashmap contains that key, it returns true else it returns false. So, let us implement it:



We have checked whether the hashmap contains the keys "China" and "Pakistan" in it or not. We get true for the key "China" as it exists in the hashmap and we get false for the key "Pakistan" as it does not exist. (refer to the output below)



Now let us study another function called keyset.

**6. Key Set in Hashmaps**

**Time Complexity:**

**O(1)**

We know that we have key-value pairs in a hashmap. We can get only the keys in a hashmap by using the keySet() function. The key set function **returns the set of all the keys in a hashmap**. We can store it in a Set container. We will study about the Set container later. For now, you can just understand that to get the Set of keys, we store it in a Set. The implementation is given below:

**Java Code (keySet() Function)**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args)

{

HashMap< String, Integer> hm = new HashMap< >();

hm.put("India", 1391);

hm.put("China", 1398);

hm.put("USA", 329);

hm.put("Indonesia", 268);

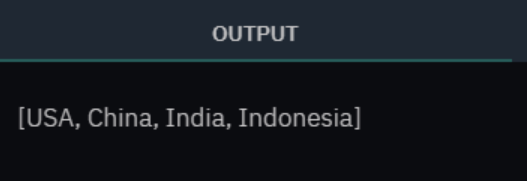
Set< String> keys = hm.keySet();

System.out.println(keys);

}

}

We get the set of keys into the set keys container. The Set keys is displayed also. (refer to the output given below)



How is this keySet() function useful to us? Well, we can apply a loop on the hashmap using this. An example for the same is shown below:

**Java Code (Implementation of keySet())**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args)

{

HashMap< String, Integer> hm = new HashMap< >();

hm.put("India", 1391);

hm.put("China", 1398);

hm.put("USA", 329);

hm.put("Indonesia", 268);

//Using keySet() to iterate through the hashmap

for (String key : hm.keySet())

{

Integer val = hm.get(key);

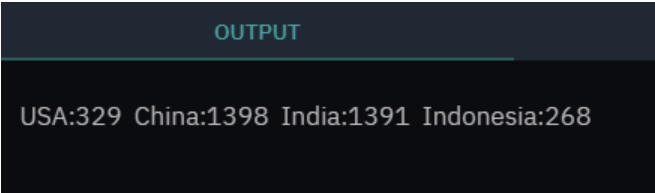
System.out.print(key + ":" + val + " ");

}

}

}

We have iterated through the hashmap using the keySet() function and we print the key value pairs. The output for the same is shown below:



So, we have seen some of the basic and most commonly used functions of the hashmap. If you notice, all the functions that we have studied have O(1) time complexity and hence hashmap becomes so important and useful to us. In many questions, it will help us reduce the time complexity and improve the performance of our code.