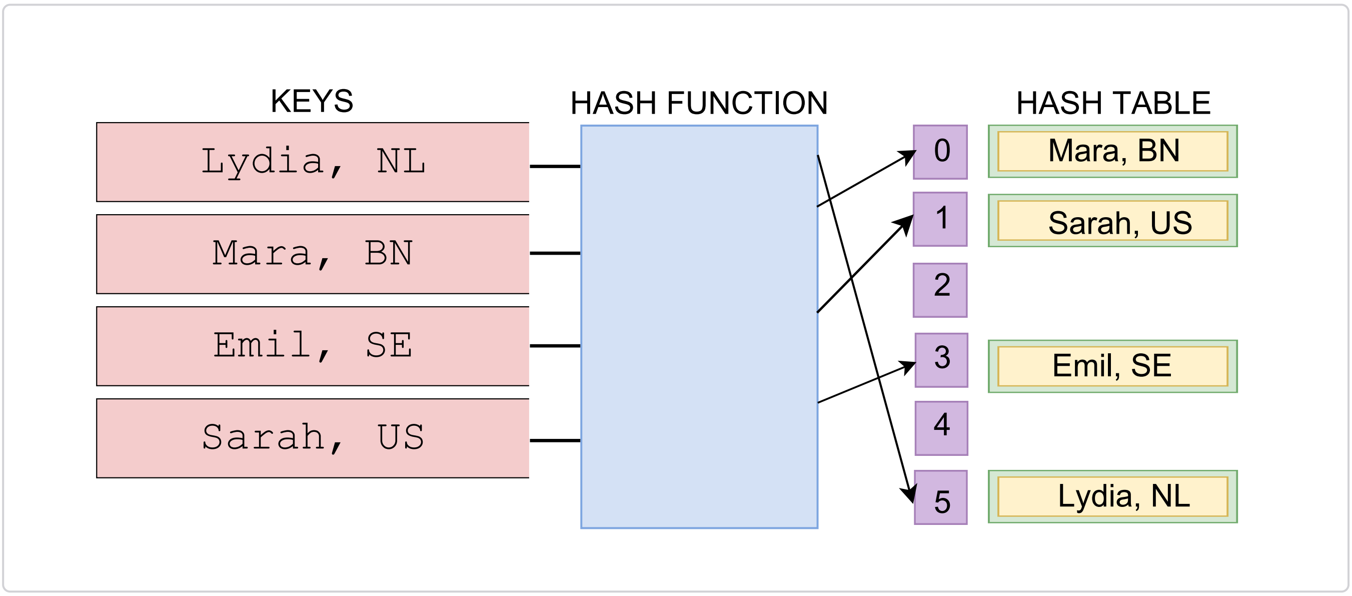
Introduction to Hash Table

A hash table is a data type that can map keys to values. (Reading time: under 3 minutes)

Hash tables are very efficient. Let’s say that we want to look up a specific person in an array: we would have to walk through every item to look for that person! The space complexity would be O(n), as space depends on the size of the array.

To look things up way more efficiently, you can use hash tables! Hash tables are made up of two parts: an **object** with the table where the data will be stored, and a **hash function** (or mapping function). This function creates a mapping by assigning the input data to a very specific index within the array! This function takes a key, and **always returns the same index for the same key**! If we would run the same key through the function twice, it gives us the same index.



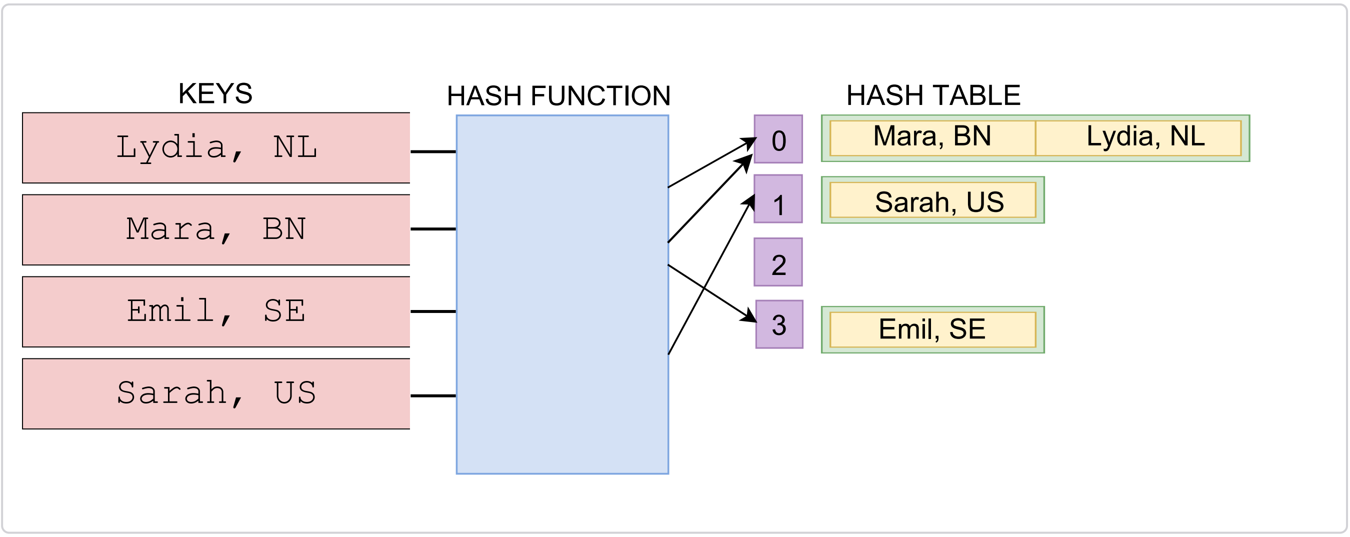
If we log this hash table (after implementing it), we get:

```
{
   values: {
      0: { "Mara": "BN" },
      1: { "Sarah": "US" },
      3: { "Emil": "SE" },
      5: { "Lydia": "NL" },
    5: { "Lydia": "NL" },
  },
  length: 4,

  size: 6,
}
```

As the hash function always gives the same hash for every key, we can easily look up key/value pairs. Instead of having to map over an entire object, we just pass the key we want to the hash function, which then returns the index where this key/value pair is stored!

However, sometimes the hash function returns the same index for different keys. This is called **collision**.



There are now two key/value pairs at hash 0. If we now want to find the key/value pair with the key “Lydia”, we first have to iterate through this bucket, until we find the right key (the bucket is shown in green).

In the next lesson, I will talk about the implementation of this data structure.