Understanding hash tables

Selecting transcript lines in this section will navigate to timestamp in the video

- Now that you understand hashing, and what a hash is, let's look at something called a hash table. A hash table is a typical way of implementing an associative array. When a hash table's created internally, it's really an array-based data structure where we add extra functionality to get us past the limitations of an array. We use the term bucket to describe each entry or place for a key-value pair to go in a hash table. We'll never add just a key or just a value. We'll always add a pair. Depending on the language, we might use the word put, add or insert to add a new key value pair. When we add a new pair to the hash table, a lot of stuff is going on behind the scenes. The key will go through a hash function and then an integer value will pop out. Depending on the function, it could be a rather large number, so we can't just use it as an array index in our implementation but we can use it to calculate an array index. This requires some logic and it's all done within the hash table implementation. Diving in, our goal is to make the integer hash value smaller but in a way that's related to the current size of the hash table. Why do this? It helps us evenly distribute the elements across the buckets we have. To keep this simple, we can use a modular operation. Essentially we take the hash value, divide it by the size of the hash table, or the number of buckets in the table and use the remainder as the index for where we'll store the value for a given key. Why's this useful? If we want to get a specific value and know the key, we can just put the key through the hash function, retrieve the hash value, run the modulo operation on it and access the specific index that the modulo outputs. Again, it's using the idea of matching hashes in order to find the correct location. There is no linear search and no traversing through a series of elements. We just go straight to the element with the key. Now sometimes two keys can have the same hash which means the values of those keys could live at a single index. We call this a collision. For example, let's say we have the value Houston in here as well as Dallas as both are large cities in Texas that have over a million residents. What's the solution? Instead of having a single value in that slot, we can create a linked list that holds both values. We call this separate chaining and it does affect performance because now to find the appropriate value, the algorithm would have to traverse the linked list in that particular slot. Accessing the individual slot would be quick but traversing each element within the linked list would take time. It's okay if this happens infrequently but we don't want it happening all the time. As the size of the bucket grows, we lose the benefits of a hash table which allow us to quickly look up a value for a particular key. Now there are other techniques for managing collisions inside hash tables but as practical programmers, we'll be using whatever's available in our current language.