Create a Hash Table

In this challenge we will learn about hash tables. A Hash table is used to implement associative arrays, or mappings of key-value pairs, like the objects and Maps we have just been studying. A JavaScript object could be implemented as a hash table, for instance (its actual implementation will depend on the environment it's running in). The way a hash table works is that it takes a key input and hashes this key in a deterministic way to some numerical value. This numerical value is then used as the actual key the associated value is stored by. Then, if you try to access the same key again, the hashing function will process the key, return the same numerical result, which will then be used to look up the associated value. This provides very efficient O(1) lookup time on average.

Hash tables can be implemented as arrays with hash functions producing array indices within a specified range. In this method, the choice of the array size is important, as is the hashing function. For instance, what if the hashing function produces the same value for two different keys? This is called a collision. One way to handle collisions is to just store both key-value pairs at that index. Then, upon lookup of either, you would have to iterate through the bucket of items to find the key you are looking for. A good hashing function will minimize collisions to maintain efficient search time.

Here, we won't be concerned with the details of hashing or hash table implementation, we will just try to get a general sense of how they work.

Let's create the basic functionality of a hash table. We've created a naive hashing function for you to use. You can pass a string value to the function `hash` and it will return a hashed value you can use as a key for storage. Store items based on this hashed value in the `this.collection` object. Create these three methods: `add`, `remove`, and `lookup`. The first should accept a key value pair to add to the hash table. The second should remove a key-value pair when passed a key. The third should accept a key and return the associated value or `null` if the key is not present.

Be sure to write your code to account for collisions!

Note: The `remove` method tests won't pass until the `add` and `lookup` methods are correctly implemented.

⚗	The `HashTable` data structure should exist.
⚗	The `HashTable` should have an `add` method.
⚗	The `HashTable` should have a `lookup` method.
⚗	The `HashTable` should have a `remove` method.
⚗	The `add` method should add key value pairs and the `lookup` method should return the values associated with a given key.
⚗	The `remove` method should accept a key as input and should remove the associated key value pair.
⚗	Items should be added using the hash function.
⚗	The hash table should handle collisions.