06. DATASTRUCTS Dictionaries.

A dictionary is a data structure that stores data as key-value pairs, such as the way a phone book stores its data as names and phone numbers. When you look for a phone number, you first search for the name, and when you find the name, the phone number is found right next to the name. **The key is the element you use to perform a search, and the value is the result of the search.**

The JavaScript Object class is designed to operate as a dictionary. In this chapter we'll use the features of the **Object** class to build a *Dictionary* class that simplifies working with a dictionary-type object. You can perform the same functions shown in this chapter using just JavaScript arrays and objects, but creating a Dictionary class makes doing the work easier and more fun.

For example, it's a lot easier to use () to reference keys rather than having to use [] notation. There is also, of course, the advantage of being able to define functions for performing collective operations, such as displaying all entries in a dictionary, rather than having to write loops in the main program to perform the same operations.

6.1. The Dictionary Class

The basis for the **Dictionary class** is the **Array** class rather than the **Object** class. Later in this chapter we are going to want to sort the keys of a dictionary, and JavaScript can't sort the properties of an Object. Keep in mind, though, that everything in JavaScript is an object, so an array is an object. We'll start our definition of the Dictionary class with this code:

Example: *Dictionary* class constructor:

```
class Dictionary {
   constructor(){
      this.dataStore = new Array();
   }
}
```

The first function to define is ***add***(). This function takes two arguments, a **key** and a **value**. The key is the index for the value element. Here is the code:

Example:Dictionary *add*() method:

```
add(key, value) {
        this.datastore[key] = value;
}
```

Next, we define the **find**() function. This function takes a key as its argument and returns the value associated with it. The code looks like this:

Example: Dictionary *find*() method.

```
find(key) {
       return this.datastore[key];
}
```

Removing a key-value pair from a dictionary involves using a built-in JavaScript function: delete. This function is part of the Object class and takes a reference to a key as its argument. The function deletes both the key and the associated value. Here is the definition of our ***remove***() function:

Example: Dictionary *remove*() method.

```
remove(key) {
       delete this.datastore[key];
}
```

Finally, we'd like to be able to **view all the key-value pairs in a dictionary**, so here is a function that accomplishes this task:

Example: Linked List *showAll*() method.

```
showAll() {
   for (var key in this.dataStore) {
        if (this.dataStore.hasOwnProperty(key)) {
          console.log(key, this.dataStore[key]);
        }
      }
 }
```

The ***keys***() function, when called with an object, **returns all the keys stored in that object.**

6.2.1. Dictionary Class Implementation.

Finally, the next code provides the definition of the *Dictionary* class up to this point.

Example: *Dictionary* class implementation.

```
class Dictionary {

   constructor(){
      this.dataStore = new Array();
   }

   add(key, value) {
        this.datastore[key] = value;
   }

    find(key) {
         return this.datastore[key];
    }

    remove(key) {
         delete this.datastore[key];
    }

   showAll() {
      for (var key in this.dataStore) {
        if (this.dataStore.hasOwnProperty(key)) {
          console.log(key, this.dataStore[key]);
        }
      }
   }
}
```

Example: A program that uses the Dictionary class is shown in the next code:

```
// main program

var pbook = new Dictionary();

pbook.add("Mike","123");
pbook.add("David", "345");
pbook.add("Cynthia", "456");

console.log("David's extension: " + pbook.find("David"));

pbook.remove("David");

pbook.showAll();
```

The output of the previous code is:

Output:

```
David's extension: 345

Mike -> 123

Cynthia -> 456
```

6.2. Auxiliary Functions for the Dictionary Class.

We can define several functions that can help in special situations. For example, it is nice to know how many entries there are in a dictionary. Here is a *count*() function definition:

Example: *count*() method.

```
count() {
  var n = 0;
  for (var key in this.dataStore) {
     ++n;
  }
  return n;
}
```

You might be wondering why the length property wasn't used for the *count*() function. The reason is that **length doesn't work with string keys**. For example:

Example:

```
var nums() = new Array();

nums[0] = 1;
nums[1] = 2;
console.log(nums.length); // displays 2

var pbook = new Array();

pbook["David"] = 1;
pbook["Jennifer"] = 2;
console.log(pbook.length); // displays 0
```

Another helper function we can use is a *clear*() function. Here's the definition:

Example: Dictionary *clear*() method.

```
clear() {
      for (var key in this.dataStore) {
         delete this.dataStore[key];
      }
}
```

The next code illustrates how the new auxiliary functions work.

Example: Using the *count*() and *clear*() functions:

```
var pbook = new Dictionary();

pbook.add("Raymond","123");
pbook.add("David", "345");
pbook.add("Cynthia", "456");

console.log("Number of entries: " + pbook.count());
console.log("David's extension: " + pbook.find("David"));

pbook.showAll();

pbook.clear();
console.log("Number of entries: " + pbook.count());
```

The output of the previous code is:

Output:

```
Number of entries: 3

David's extension: 345

Raymond -> 123
David -> 345
Cynthia -> 456

Number of entries: 0
```

6.3. Adding Sorting to the Dictionary Class

The primary purpose of a dictionary is to retrieve a value by referencing its key. The actual order that the dictionary items are stored in is not a primary concern. However, many people like to see a listing of a dictionary in sorted order. Let's see what it takes to display our dictionary items in sorted order.

Example: Arrays can be sorted. For example:

```
var a = new Array();

a[0] = "Mike";
a[1] = "David";

console.log(a); // displays Mike, David

a.sort();

console.log(a); // displays David, Mik
```

We can't perform the same test with string keys, however. The output from the program is empty. This is much the same problem we had earlier trying to define a **count**() function. This isn't really a problem, however. All that matters to the user of the class is that when the dictionary's contents are displayed, the results are in sorted order. We can use the ***Object.keys()*** function to solve this problem. Here is a new definition for the *showAll*() function:

Example: Arrays can be sorted. For example:

```
showAll() {
   for each (var key in Object.keys(this.datastore).sort()) {
      print(key + " -> " + this.datastore[key]);
   }
}
```

The only difference between this definition of the function and our earlier definition is we've added a call to *sort*() after we obtain the keys from the datastore array. The next example demonstrates how this new function definition is used to display a sorted list of names and numbers.

Example: A sorted dictionary display.

```
var pbook = new Dictionary();

pbook.add("Raymond","123");
pbook.add("David", "345");
pbook.add("Cynthia", "456");
pbook.add("Mike", "723");
pbook.add("Jennifer", "987");
pbook.add("Danny", "012");
pbook.add("Jonathan", "666");

pbook.showAll();
```

Here is the output of the program:

Output:

```
Cynthia -> 456
Danny -> 012
David -> 345
Jennifer -> 987
Jonathan -> 666
Mike -> 723
Raymond -> 123
```