Practice exercise 3.1

1. Create an array to use as your shopping list with 3 items: "Milk," "Bread," and "Apples."
2. Check your list length in the console.
3. Update "Bread" to "Bananas."
4. Output your entire list to the console.

Array methods

We have just seen the built-in length property. We also have a few built-in methods. Methods are functions on a certain object. Instead of holding a value, like properties, they perform actions. We will cover functions in-depth in *Chapter 6, Functions*. For now, all you need to know is that you can call methods and functions, and when you do, some code that is specified inside that function gets executed.

We just accidentally saw we could add elements using new indices. This is not the proper way to do it as it is easy to make mistakes and accidentally overwrite a certain value or skip a certain index. The right way is to do this with a special method. Similarly, we can also delete elements and sort the elements in the array.

Adding and replacing elements

We can add elements with the `push()` method:

```
favoriteFruits = ["grapefruit", "orange", "lemon"];
favoriteFruits.push("tangerine");
```

The value gets added to the end of the array. The `push` method returns the new length of the array, four in this case. You can store this length in a variable like this:

```
let lengthOfFavoriteFruits = favoriteFruits.push("lime");
```

The value 5 gets stored in the `lengthOfFavoriteFruits` variable. If we log our array, `favoriteFruits`, like this:

```
console.log(favoriteFruits);
```

Here is the new array:

```
[ 'grapefruit', 'orange', 'lemon', 'tangerine', 'lime' ]
```

The new order will be:

```
[ 'Maria', 'James', 'Fatiha', 'Bert', 'Alicia' ]
```

Practice exercise 3.2

1. Create an empty array to use as a shopping list.
2. Add `Milk`, `Bread`, and `Apples` to your list.
3. Update "`Bread`" with `Bananas` and `Eggs`.
4. Remove the last item from the array and output it into the console.
5. Sort the list alphabetically.
6. Find and output the index value of `Milk`.
7. After `Bananas`, add `Carrots` and `Lettuce`.
8. Create a new list containing `Juice` and `Pop`.
9. Combine both lists, adding the new list twice to the end of the first list.
10. Get the last index value of `Pop` and output it to the console.
11. Your final list should look like this:

```
["Bananas", "Carrots", "Lettuce", "Eggs", "Milk", "Juice",
"Pop", "Juice", "Pop"]
```

Multidimensional arrays

Earlier, we established already that arrays can contain any data type. This means that arrays can also contain other arrays (which, in turn, can contain… other arrays!). This is called a multidimensional array. It sounds complicated, but it is just an array of arrays: a list of lists:

```
let someValues1 = [1, 2, 3];
let someValues2 = [4, 5, 6];
let someValues3 = [7, 8, 9];

let arrOfArrays = [someValues1, someValues2, someValues3];
```

So, we can create an array of already existing arrays. This is called a two-dimensional array. We can write it like this:

```
let arrOfArrays2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
```

If you want to access elements of the inner arrays, you will have to specify an index twice:

```
let value1 = arrOfArrays[0][1];
```

The statement will grab the first array because it has an index position of `0`. From this first array, it will take the second value, because it has an index position of `1`. Then it stores this value in `value1`. That means the value of `value1` will be 2. Can you figure out what the value of the next one will be?

```
let value2 = arrOfArrays[2][2];
```

It takes the third array, and from this third array, it takes the third value. Thus, 9 will be stored in `value2`. And it does not stop here; it can go many levels deep. Let's show that by creating an array of our array of arrays. We are simply going to store this array three times in another array:

```
arrOfArraysOfArrays = [arrOfArrays, arrOfArrays, arrOfArrays];
```

This is what the array looks like in terms of values:

```
[
  [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ],
  [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ],
  [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
]
```

Let's get the middle element of this array, which is the value 5, belonging to the second array of arrays. It is done like this:

```
let middleValue = arrOfArraysOfArrays[1][1][1];
```

The first step is to get the second array of arrays, so index 1. Then we need to get the second array of this one, which again is index 1. Now we reach the level of the values, and we need the second value, so again we use index 1. This is useful in many situations, for example, when you want to work with matrices.

Practice exercise 3.3

1. Create an array containing three values: 1, 2, and 3.
2. Nest the original array into a new array three times.
3. Output the value 2 from one of the arrays into the console.

```
  breed: 'chihuahua',
  age: 3,
  burglarBiter: true
}
```

It's useful to note that if we change the data type of one of our properties, for example:

```
dog["age"] = "three";
```

This is not a problem. JavaScript will just change the whole value and data type to the new situation.

Another element to note is that we are now using the literal string values to refer to the object's properties, but we can also work with variables to achieve this. So, for example:

```
let variable = "age";
console.log(dog[variable]);
```

This will still output `three`, as we just changed the value of age to three. If we change the value of the variable to another dog property, we will be accessing another property, like this:

```
variable = "breed";
console.log(dog[variable]);
```

This will print `chihuahua`. And when we update the value accessing this way, it is the exact same as when we would have accessed it with the literal string:

```
dog[variable] = "dachshund";
console.log(dog["breed"]);
```

So, this will log `dachshund` to the console.

Practice exercise 3.4

1. Create a new `myCar` object for a car. Add some properties, including, but not limited to, `make` and `model`, and values for a typical car or your car. Feel free to use booleans, strings, or numbers.

```
"manufacturing toys"],
                    address: [{
                      street: "2nd street",
                      number: "123",
                      zipcode: "33116",
                      city: "Miami",
                      state: "Florida"
                    },
                    {
                      street: "1st West avenue",
                      number: "5",
                      zipcode: "75001",
                      city: "Addison",
                      state: "Texas"
                    }],
                    yearOfEstablishment: 2021
                  };
```

To access elements of increasingly nested objects and arrays, we simply extend the same logic you have seen in the previous sections. To access the street name of Healthy Candy's first address, we can use the following code:

```
let streetName = company.address[0].street;
```

As you see, we can stack object and array element requests indefinitely.

We will not make it any more complicated than this for now. Whenever you need a list of something, you will be using an array. Whenever you want to represent something with properties that have descriptive names, it is better to use an object. Just remember that object properties can be of any type.

Practice exercise 3.5

1. Create an object named `people` that contains an empty array that is called `friends`.
2. Create three variables, each containing an object, that contain one of your friend's first names, last names, and an ID value.
3. Add the three friends to the `friend` array.
4. Output it to the console.

Chapter projects

Manipulating an array

Take the following array:

```
const theList = ['Laurence', 'Svekis', true, 35, null, undefined,
{test: 'one', score: 55}, ['one', 'two']];
```

Manipulate your array using various methods, such as `pop()`, `push()`, `shift()`, and `unshift()`, and transform it into the following:

```
["FIRST", "Svekis", "MIDDLE", "hello World", "LAST"]
```

You can take the following steps, or adopt your own approach:

- Remove the first item and the last item.
- Add `FIRST` to the start of the array.
- Assign `hello World` to the fourth item value.
- Assign `MIDDLE` to the third index value.
- Add `LAST` to the last position in the array.
- Output it to the console.

Company product catalog

In this project, you will implement a data structure for a product catalog and create queries to retrieve data.

1. Create an array to hold an inventory of store items.
2. Create three items, each having the properties of name, model, cost, and quantity.
3. Add all three objects to the main array using an array method, and then log the inventory array to the console.
4. Access the quantity element of your third item, and log it to the console. Experiment by adding and accessing more elements within your data structure.

Self-check quiz

1. Can you use `const` and update values within an array?
2. Which property in an array gives the number of items contained in the array?
3. What is the output in the console?

```
const myArr1 = [1,3,5,6,8,9,15];
console.log(myArr1.indexOf(0));
console.log(myArr1.indexOf(3));
```

4. How do you replace the second element in an array `myArr = [1,3,5,6,8,9,15]` with the value 4?
5. What is the output in the console?

```
const myArr2 = [];
myArr2[10] = 'test'
console.log(myArr2);
console.log(myArr2[2]);
```

6. What is the output in the console?

```
const myArr3 = [3,6,8,9,3,55,553,434];
myArr3.sort();
myArr3.length = 0;
console.log(myArr3[0]);
```

Summary

So, in this chapter, we have seen arrays and objects. Arrays are a list of values. These could be values of the same type, but also values of different types. Every element of the array gets an index. The index of the first element is 0. We can access the elements of the array using this index. We can also use this index to change and delete the element.

We then saw that it is also possible to have arrays containing other arrays; these are multidimensional arrays. To access the elements of a multidimensional array, you would need to use as many indices as you have nested arrays.

Then, we covered objects and learned that arrays are a special kind of object. Objects contain properties and methods. We looked at the properties of objects and saw that these properties are given a name and can be accessed and modified using this name.

We ended this module by looking at how arrays can contain objects, and how objects can contain arrays and more. This enables us to create complex object structures, which will be of great use in designing real-life applications.