Hyperiondev

TASK

Javascript IV: Arrays

Visit our website

Introduction

WELCOME TO THE FOURTH JAVASCRIPT TASK!

In this task, you will learn how to work with arrays. Arrays are very important data structures that allow you to store several values in one variable. You will also learn about sets and maps, two more data structures that are similar to arrays.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHAT IS AN ARRAY?

An array is a collection of related data. For example, we could create an array of student marks as follows:

```
let studentMarks = [10, 40, 80, 99];
```

Let's pay close attention to the syntax of how we declare an array. The variable `studentMarks` is no different from the other variables we've been declaring all along. This code has created an array with four values, each separated by commas, which is then assigned to the variable called `studentMarks`.

You can access a specific value stored in an array by using its position in the array or *index*. Like in strings, the first position in an array is always 0. In order to retrieve an item, simply write the array variable's name, followed by the position of the value you would like between square brackets, [ ].

For example, here's how to access the first element of the array:

```
studentMarks[0];
```

Of course, doing the above would not assign the value of the array at position zero to a variable or output it to the console or anything similar without a little more code.

LOOPING THROUGH AN ARRAY

Often we need to visit each element in the array and perform a uniform operation on it. We can do this by looping through the array. There are several types of loops we can use to loop through an array. The most important of these are discussed in this task.

Using a *for loop*

Remember our discussion of the difference between a *for loop* and a *while loop*? A *for loop* is more appropriate to use when you know how many times you want to run through the loop. As such, *for loops* are perfect to use with arrays because we know exactly how many elements there are in the array. So let's write a *for loop* that will run through the array and print out each item as a bulleted list.

The first thing we do is determine the length of the array. We do this using the `length()` function. This function will return an integer that tells us how many items

there are in our array. In the example below, we store this result in a variable called **arrayLength**:

```
arrayLength = studentMarks.length;
```

We won't always know how many items there are in the array as we code so this function comes in handy. Note that the length function doesn't take any arguments.

Next, we need to write a *for loop* that will iterate from the beginning of this array until the end:

```
for (i = 0; i < arrayLength; i++) {
    text += studentMarks[i];
}
```

As per convention, we use a variable called *i* as the control variable. Note that we initialise it to 0. The termination condition will evaluate to false as long as *i* is less than the length of the array, stored in **arrayLength**. Note that the index values of the array will start at zero and run to one less than the length of the array (as you will recall from an earlier lesson, the length is always one more than the final index value). Finally, we increment *i* at the end of each iteration of the loop.

In the body of the *for* loop we are accessing an element of the array; the square brackets are an indication of this. But which element are we accessing? Well, that depends on the value inside the square brackets. Remember that *i* is our control variable. It will start with the value of 0, which is then incremented by 1 each time the loop runs. Therefore, in the course of its run for the loop above, *i* will assume the values 0, 1, 2, and 3 before the *for loop* terminates. Note that 0, 1, 2, and 3 also happen to be the indexes we need to reference all elements in our array. Let's draw a trace table to illustrate what will happen in each iteration of the loop:

Iteration of the loop	i	studentMarks[i]	Value Accessed
1st	0	studentMarks[0];	10
2nd	1	studentMarks[1];	40
3rd	2	studentMarks[2];	80
4th	3	studentMarks[3];	99

So when the value of *i* is 0, the first element will be accessed. When the value changes in the second cycle to 1, the second element is accessed. This pattern will repeat until all the numbers have been printed in a list.

All the other loops that we can use to loop through an array work in basically the same way as the *for loop* but the syntax we use differs. Another loop we can use to loop through arrays is the *for...of* loop. Let's find out a bit more abut this.

Using a *for of* Loop

The *for...of* loop allows you to loop through any iterable object including strings, arrays, and objects. An example of a *for...of* loop is shown below. See how this loop is used to loop through all the elements in the array called `nums`. If you need more help with *for...of* loops, see **here**. The *for...of* loop is a new JavaScript loop that was introduced with ES6.

```
let nums = [10, 20, 30];
for (let value of nums) {
  console.log(value);
}
```

Using the *forEach* method

JavaScript provides a special method that makes it easier to loop through an array. The `forEach()` method executes a provided function once for each array element. See the example below that illustrates how the *forEach* method works. Notice that the *forEach* method in the example below takes an anonymous function (a function without a name) as an argument. The *forEach* method will execute the instructions found in that function for each element in the array.

```
let nums = [10, 20, 30];

nums.forEach(function(element) {
  console.log(element);
});
```

OTHER ARRAY METHODS

Here are a few more methods you can use to manipulate arrays other than just looping through them:

- Add an item to the array at the end position using the *push* method. E.g.

```
let nameList = ["James", "Molly", "Chris", "Peter", "Kim"];
 nameList.push("Tom")
```

- Find the index of an element in an array using the *indexOf* method. E.g.

```
let names =["Smith", "Adams", "Nkanjeni"];
names.indexOf("Adams");
```

The code above would return the value 1 since the first index of an array is always 0.

- Make a copy of an array using the *slice* method. E.g.

```
let copyOfNames = names.slice();
```

- Changing an element in a list:

```
let nameList = ["James", "Molly", "Chris", "Peter", "Kim"];
nameList[2] = "Tom";
```

The value at the third position changes from 'Chris' to 'Tom'.

- Deleting an element in a list:

```
let charList = ['J', 'a', 'v', 'a', 's', 'c', 'r', 'i', 'p', 't'];

charList.pop();     //['J', 'a', 'v', 'a', 's', 'c', 'r', 'i', 'p']
charList.shift();   //['a', 'v', 'a', 's', 'c', 'r', 'i', 'p']
delete charList[1]; //['a', undefined, 'a', 's', 'c', 'r', 'i', 'p']
charList.splice(1,1); //['a', 'a', 's', 'c', 'r', 'i', 'p']
```

The `splice()` function takes a starting position as the first argument and a the number and the number of elements to remove as the second argument - in this case, start at index position 1 and remove 1 element, the undefined element. Look at **this** for a more in-depth explanation of how the `splice()` function works and exactly what is happening in the last line of code in the example.

JAVASCRIPT ARRAY METHODS

There are many useful built-in array methods available for you to use. We have already looked at a few.

Some of the most useful array methods are listed below:

- *indexOf()* - Searches the array for the given element and returns its position
- *includes()* - Checks if the array contains a given element
- *remove()* - Removes an item from the list
- *pop()* - Removes the last element in the array
- *push()* - Adds an element to the end of the array and returns the new length
- *shift()* - Removes the first element in the array
- *sort()* - Sorts elements in the array in ascending order
- *splice()* - Adds or removes elements in an array
- *reverse()* - Reverses the order of elements in the array

SPOT CHECK 1

Let's see what you can remember from this section.

1. Fill in the blanks:
   a.
   ```
   let nums = [10, 20, 30];
   for (let ____ of ____) {
     console.log(value);
   ```
   b.
   ```
   let nums = [10, 20, 30];
   ____.____(function(____) {
     console.log(element);
   });
   ```
2. What does the *push* method do?
3. What does the *indexOf* method do?
4. What does the *slice* method do?
5. Which method would you use to remove the last element of the array?
6. Which method would you use to remove the first element of the array?

SETS

ES2015 introduces a new object that we can use. A set is like an array in that it stores a collection of items. It is different from an array in a number of important ways though:

- It stores only unique/distinct items. A set will, therefore, not store any duplicate values but an array will.
- A set is not indexed.
- Items in a set can't be accessed individually.

```
const mySet = new Set([1, 2, 3, 4, 5, 5, 5]);
```

```
console.log(mySet); // 1, 2, 3, 4, 5 duplicates would be removed
```

Think about it:

- If you were storing student marks would you use an array or a set? Why?

MAPS

Maps are similar to sets, but instead of storing a collection of individual items, the items contained in a map are key-value pairs. When the key is known, you can use it to retrieve the value associated with it, i.e. each key *maps* to a value and can be used to access it. The code example below from **here** shows how you can use maps.

```
let myMap = new Map();
myMap.set(0, 'zero');
myMap.set(1, 'one');
for (let [key, value] of myMap) {
  console.log(key + ' = ' + value);
}
// 0 = zero
// 1 = one

for (let key of myMap.keys()) {
  console.log(key);
}
// 0
// 1

for (let value of myMap.values()) {
  console.log(value);
}
// zero
// one

for (let [key, value] of myMap.entries()) {
  console.log(key + ' = ' + value);
}
// 0 = zero
// 1 = one
```

JAVASCRIPT MAP METHODS

There are many useful built-in map methods available for you to use. We have already looked at *set()*.

Some other map methods can be found below:

- *has()* - checks to see if a key exists in the map and returns a boolean
- *get()* - returns the value paired with a given key
- *delete()* - deletes the given key-value pair from the map
- *clear()* - removes all key-value pairs in a map

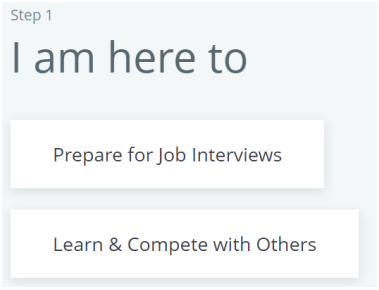
SPOT CHECK 2

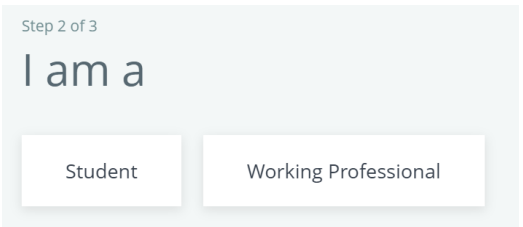
Let's see what you can remember from this section.

1. What are the main differences between a set and an array?
2. What is unique about maps?
3. What does the *get* method do in relation to maps?
4. What does the *has* method do in relation to maps?

Extra resource

HackerRank is an excellent resource for newbies and professional coders alike. It is a hub of standardised coding exercises that you can use to practise your skills as you go through this bootcamp. Go to **https://www.hackerrank.com/** and sign up. Once you have created an account, select "I am here to prepare for job interviews", "I am a student" and fill in when you are expected to graduate from HyperionDev.





Once you're on your dashboard, scroll down to "Skills Available For Practice" and select Algorithms — you will then be able to select JavaScript as your coding language when

you open your first challenge. You will be taken to a whole bunch of algorithmic challenges to try. You should already be able to solve a couple, but keep checking back to your dashboard as you go through the bootcamp to see how you progress!

HackerRank is a very common platform for companies on which to interview and test potential developers, so make sure that you are comfortable with the way that problems are phrased and how the platform works.

Instructions

Open **example.js** in VSCode and read through the comments before attempting these tasks.

Getting to grips with JavaScript takes practice. You will make mistakes in this task - this is completely to be expected as you learn the keywords and syntax rules of this programming language. It is vital that you learn to debug your code. To help with this remember that you can:

- Use either the JavaScript console or Sublime Text (or another editor of your choice) to execute and debug JavaScript in the next few tasks.
- Remember that if you really get stuck you can contact a code reviewer for help.

Compulsory Task 1

Follow these steps:

- Create a .js file called **arrayTask.js** and open **arrayTask.html**. Feel free to modify **arrayTask.html** as needed for this task.

- In **arrayTask.js** write the code needed to do the following (use functions):
    - Create an array called 'favLanguages' that includes at least 5 programming languages that you would like to learn. Use a loop to display each programming language in 'favLanguages' on **arrayTask.html** in the list element: `<ul id="favLanguages">`.

    - Create an array called 'myJSTestResults' that stores a list of results out of 100 that you have scored for 5 fictitious tests that you have written in a fictitious JavaScript class.
      E.g. `let myJSTestResults = [40, 60, 80, 80, 85];`

    - Write the code that will calculate the average grade based on the results in my 'myJSTestResults'. For example, the average grade based on the values in the example above would be:
      `let average = (40 + 60 + 80 + 80 + 85)/5;`
      Based on the average grade, decide which letter symbol should be assigned. Use the table below:

Average percentage	Letter symbol
80 - 100	A
70 - 79	B
60 - 69	C
50 - 59	D
49 or less	F

- Output your results to the div element with the id "myGrades" in **arrayTask.html**.

Compulsory Task 2

Follow these steps:

- Create a .js file called **task2.js** and a file called **task2.html**. Write the code necessary to do the following:
    - Use a map to store the names of 5 students with their corresponding average grade. You can initialise this map with fictitious hardcoded names and grades.
    - Write the JavaScript to display the name of each student in a drop-down menu. When the student's name is selected from the drop-down menu, the student's grade should be displayed using an alert. (Hint: Remember that you can use the value and innerHTML attributes of an `<option>` element.)

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Completed the task(s)?

Ask an expert to review your work!

Review work

Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Visual Studio Code** may not be installed correctly.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

SPOT CHECK 1 ANSWERS

1.
    a. `value, nums`
    b. `nums, forEach, element`
2. The *push* method adds an element to the end of the array and returns the new length
3. The *indexOf* method searches the array for the given element and returns its position
4. The *slice* method makes a copy of an existing array.
5. The *pop* method.
6. The *shift* method.

SPOT CHECK 2

1. What are the main differences between a set and an array?
    a. A set will not store any duplicate values but an array will.
    b. A set is not indexed.
    c. Items in a set can't be accessed individually.
2. Maps store key-value pairs instead of single values like sets and arrays.
3. The *get* method returns the value of the given key
4. The *has* method checks to see if a key exists in the map and returns a boolean