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 your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor 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:

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

Let's play 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*. 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];
```

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** 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. **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 bullet 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.

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 += "<li>" + studentMarks[i] + "</li>";
}
```

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**. 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 bracket. 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 (trace tables are used by programmers to track the values of variables as they change throughout the program) to illustrate what will happen in each iteration of the loop:

Iteration of the loop	i	studentMarks[i]	Value Accessed
1	0	studentMarks[0];	10
2	1	studentMarks[1];	40
3	2	studentMarks[2];	80
4	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 run 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** statement.

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);
}
```

A note from our coding mentor
Nkosi

ES6 Update: No more 'var'

Look carefully at the example of the for...of loop above. Notice that the variables are declared with the keyword "let" instead of the normal "var" keyword we have used so far. With previous versions of ECMAScript, you could only declare variables with the keyword "var". With ES2015 (or ES6), variables are declared with either the keywords "let" or "const". Variables declared with the keywords, let or const are scoped within the code block they are declared in. Variables declared using the keyword var, on the other hand, were declared either locally or globally in relation to a function (which sometimes lead to errors). const variables must be assigned a value when they are declared and the value they are assigned cannot change. let variables can be re-assigned a value within the block where they have been declared but they cannot be redeclared.
In summary, with ES2015:

- Don't use var any more.
- Use const whenever possible to initialize variables

- Use let whenever the value of that variable needs to be changed in a block of code.

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

Besides looping through an array, here are a few more methods you can use to manipulate an array:

- Add an item to the array the end of an array using the *push* method. E.g. nums.push(40); would add the element 40 to an array called nums.
- 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();
```

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
```

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. The code example below from here shows how you can use maps.

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

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

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

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

Instructions

Open *example.js* in Sublime Text 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 your mentor 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. var 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:

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.)

Once you have completed the task in line with the instructions above, click the button below to request your mentor to review your work and provide feedback. If you have any questions while attempting the task, leave your mentor a note on the comments.txt file in your Student Dropbox folder.

Completed the task(s)?

Ask your mentor review your work!

Review work

Things to look out for:

1. Make sure that you have installed and setup all programs correctly. Follow the instructions in the FAQs document that accompanies this task to configure Sublime Text for debugging and executing JavaScript.

Rate us

Share your thoughts

Hyperion 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.