TASK

Beginner Data Structures — Arrays and Maps

Visit our website

Introduction

WELCOME TO THE BEGINNER DATA STRUCTURES — ARRAYS AND MAPS TASK!

This task aims to ensure that you have a concrete understanding of strings and array manipulation, and also to give you a little introduction to maps. In **example.js**, you will see examples that deal with operations that can be applied to elements in arrays as well as maps. This task also touches on a few built-in functions and how they can be used to compute certain values on array elements as well as maps.

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 expert code reviewers are 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];
```

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 this reason, *for* loops are perfect to use with arrays because we know exactly how many elements there are in the array. 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`:

```
let 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 (let i = 0; i < arrayLength; i++) {
    console.log(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`**. 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
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);
}
```

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 at the end of an array 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"];
let nameIndex = names.indexOf("Adams");
console.log(nameIndex);
```

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";
```

- 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']
```

JAVASCRIPT ARRAY METHODS

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

Some other array methods can be found 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

SETS

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

- 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. When the key is known, you can use it to retrieve the value associated with 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 of the given key
- *delete()* - deletes the given key-value pair from the map
- *clear()* - removes all key-value pairs in a map

Instructions

Open **example.js** in Visual Studio Code 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 Visual Studio Code (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 an expert code reviewer for help.

Compulsory Task 1

Follow these steps:

- *Note: For this task you will need to create an HTML file to get input from a user. If you need a refresher on how to do this, go back to the* ***example.js*** *and* ***index.html*** *files in your Task 2 folder for a refresher.*
- Create a new JavaScript file in this folder called **guestList.js**.
- Ask the user to input names of people they would like to invite to a dinner party.
- Each name should be added to an array.
- The user can only invite a maximum of ten people. If they try to add more than 10 names, the program should state, “You have already added 10 people to your guest list.”
- The program should then output the list.

Compulsory Task 2

Follow these steps:

- You are now going to add more functionality to **guestList.js**.
- Now, if the user tries to add an 11th name to the array, the program should say, “You have already added 10 people to your guest list. Would you like to replace someone on the list with this person? y/n: ”
- If the user enters “y”, the program outputs the current guest list and then asks, “Who would you like to replace?”
- The program then replaces the person the user specifies with the 11th person given and outputs the updated list.

Compulsory Task 3

Follow these steps:

- Create a new JavaScript file in this folder called **translator.js**.
- You are going to create a map that can be used as a translator for a language of your choice.
- Create a map with 10 key-value pairs, where the English word is the key, and the translated word is the value.
- Ask the user what word they would like to translate.
- When the user inputs a word that is one of the keys, it should output the corresponding value.

Rate us

Share your thoughts

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

Think that 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.