TASK

Data Structures - Arrays, Strings, and Maps

Visit our website

Introduction

WELCOME TO THE DATA STRUCTURES - ARRAYS, STRINGS, AND MAPS TASK!

This task aims to ensure that you have a concrete understanding of strings and arrays and how to manipulate them. It also aims to provide an 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 methods, and how they can be used to compute certain values on array elements as well as maps.

WHAT IS AN ARRAY?

An array is a collection of related data. In the example below we have created an array of student marks. This array stores four values, each separated by commas, which are then assigned to the array variable called `studentMarks`. You can access a specific value stored in an array by using its position in the array, also called the *index*. The first position in an array is always zero (0). In order to return the value of the first item in the console, simply use the `console.log` method. Within this method you can enclose the array variable's name, followed by the position of the value (i.e. the index value) you would like between square brackets `[ ]`.

```
let studentMarks = [10, 40, 80, 99];
console.log(studentMarks[0]); // Output: 10
```

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.

Using a `for` 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 by checking the array's length, we can easily determine exactly how many elements there are in the array. Let's write a `for` loop that will run through an array and print out each item as a bulleted list.

The first thing we need to do is determine the length of the array. We do this using the `.Length` property of the array object. This property will return an integer that

tells us how many items are in our array. We will store this result in a variable called **arrayLength**. We won't always know how many items there are in the array as we code, so this property comes in handy. Next, we need to write a for loop that will iterate from the beginning of this array until the end, as follows:

```
let studentMarks = [10, 40, 80, 99];
let arrayLength = studentMarks.length;
for (let i = 0; i < arrayLength; i++) {
  console.log(studentMarks[i]);
}
// 10
// 40
// 80
// 99
```

Conventionally 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, as indicated by the square brackets. 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

As you can see, 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 continue 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 approach we can

use to loop through arrays is using a `for...of` statement. This is new to you, so let's take a closer look.

Using a `for of` Loop

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

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

Using the `forEach` method

The `forEach` method is another array-iteration approach. It executes a provided function once for each array element. The example below illustrates how the `forEach` method 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 numbers = [10, 20, 30];
numbers.forEach(function (element) {
  console.log(element);
});
// 10
// 20
// 30
```

BUILT-IN JAVASCRIPT METHODS

Besides looping through an array, here are a few more methods you can use to manipulate the elements in an array.

Array methods

Arrays are integral to almost every JavaScript program, and JavaScript's built-in array methods allow us to manipulate and interact with arrays in numerous ways. Here are some of the most commonly used array methods:

`push()`: This method adds one or more elements to the end of an array and returns the new length of the array:

```
let movies = ["Harry Potter"];
movies.push("Lord of the Rings", "Star Wars");/* this returns 3 but we don't capture the value anywhere in this example*/
console.log(movies); // Output: [ 'Harry Potter', 'Lord of the Rings', 'Star Wars' ]
```

`pop()`: This method removes the last element from an array and returns that element:

```
let movies = ["Harry Potter", "Lord of the Rings", "Star Wars"];
let lastMovie = movies.pop();
console.log(lastMovie); // Output: Star Wars
```

`shift()`: This method removes the first element from an array and returns that element. This method changes the length of the array:

```
let movies = ["Harry Potter", "Lord of the Rings", "Star Wars"];
let firstMovie = movies.shift();
console.log(firstMovie); // Output: Harry Potter
```

`unshift()`: The `unshift()` method adds new items to the beginning of an array, and returns the new length:

```
let movies = ["Harry Potter", "Lord of the Rings"];
movies.unshift("Star Wars", "Avatar");/* this returns 4 but we don't capture the value anywhere in this example*/
console.log(movies); // Output: [ 'Star Wars', 'Avatar', 'Harry Potter', 'Lord of the Rings' ]
```

`sort()` and `reverse()`: The sort method sorts elements in ascending order, whereas the reverse method reverses the order of elements in the array:

```
let movies = ["Star Wars", "Avatar", "Harry Potter", "Lord of the Rings"];
movies.sort();
```

```
console.log(movies);  //  Output:  [  'Avatar',  'Harry  Potter',  'Lord  of  the
Rings', 'Star Wars' ]

movies.reverse();
console.log(movies);  //  Output:  [  'Star  Wars',  'Lord  of  the  Rings',  'Harry
Potter', 'Avatar' ]
```

splice(): This method adds or removes elements in an array:

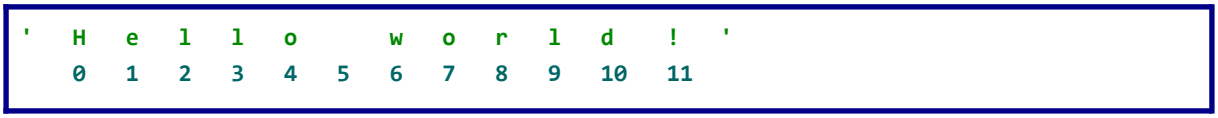
```
const movies = ["Superman", "Spiderman", "Thor", "Wonder Woman"];
// Starting at position 1, remove 2 elements of the array
movies.splice(1, 2);
console.log(movies); // Output: [ 'Superman', 'Wonder Woman' ]

/* 2 new movies inserted amongst the existing movies at position 1. The 0
means no movies are overwritten/removed. THe initial entry at position 1
(Wonder Woman) is just moved up 2 places to make room for the insertions */
movies.splice(1, 0, "Batman", "The Flash");
console.log(movies); // Output:  [ 'Superman', 'Batman', 'The Flash', 'Wonder
Woman' ]

/* Insert 2 new movies starting at and overwriting position 2. Even though 2
movies are inserted, position 3 will not be overwritten unless the 1 is
changed to a 2 */
movies.splice(2, 1, "Alien", "Star Trek");
console.log(movies); // Output: [ 'Superman', 'Batman', 'Alien', 'Star Trek',
'Wonder Woman' ]

/* Insert 2 new movies starting at and overwriting position 2 and replacing 3
entries. As there are only 2 movies being inserted this adds them in place of
Alien and Star Trek, and removes the movie at the next position (Wonder Woman)
from the list */
movies.splice(2, 3, "The Godfather", "Pulp Fiction");
console.log(movies); // Output: [ 'Superman', 'Batman', 'The Godfather', 'Pulp
Fiction' ]
```

STRINGS

One of the most crucial concepts to grasp when it comes to programming is string handling. In this section, we will learn more about the different built-in JavaScript methods we can use to manipulate strings in order to create more advanced programs.

INDEXING STRINGS

You can think of the string '`Hello world!`' as a list, and each character in the string as an item with a corresponding index.



The space and the exclamation point are included in the character count, so '`Hello world!`' is 12 characters long, from '`H`' at index 0 to '`!`' at index 11.

String methods

String manipulation is a key aspect of many programming tasks, and JavaScript provides many methods for string data. Some common ones include:

`toUpperCase()`: This method returns the calling string value converted to uppercase:

```
let greeting = "Hello, World!";
let upperCaseGreeting = greeting.toUpperCase();
console.log(upperCaseGreeting); // Output: 'HELLO, WORLD!'
```

`toLowerCase()`: This method returns the calling string value converted to lowercase:

```
let greeting = "Hello, World!";
let lowerCaseGreeting = greeting.toLowerCase();
console.log(lowerCaseGreeting); // Output: 'hello, world!'
```

`slice()`: This method extracts a section of a string and returns it as a new string, without modifying the original string. You can slice strings by specifying a range from one index to another. **Note: the starting index is *included* and the ending index is *excluded*.**

```
let string = "The morning is upon us.";
let stringSlice = string.slice(4, 11);
console.log(stringSlice); // Output: 'morning'
```

split() and join(): The split method splits a string into individual substrings, whereas the join method joins two strings together using a separator:

```
const shoppingList = "chocolate juice popcorn muffins";
const newList = shoppingList.split(" ").join("\n");
console.log(newList);
// Output:
// chocolate
// juice
// popcorn
// muffins
```

Note: There are also methods which you could use for both arrays and strings:

indexOf(): This method can return the index of the first occurrence of the specified value within the array or string. If the item is not found, -1 will be returned.

```
const moviesArray = ["Star Wars", "Avatar", "Harry Potter", "Lord of the
Rings"];

let movieIndex = moviesArray.indexOf("Avatar");
console.log(movieIndex); // Output: 1

let string = "The quick brown fox jumps over the lazy dog.";
let wordIndex = string.indexOf("fox");
console.log(wordIndex); // Output: 16
```

includes(): This method checks if the array contains a given element:

```
let moviesArray = ["Star Wars", "Avatar", "Harry Potter", "Lord of the
Rings"];
if (moviesArray.includes("Harry Potter")) {
  console.log("This movie is in the array.");
} else {
  console.log("This movie is not in the array.");
} // Output: This movie is in the array.

let string = "The quick brown fox jumps over the lazy dog.";
```

```
let wordIndex = string.includes("cat");
console.log(wordIndex); // Output: false
```

Take note:

Through understanding and applying these built-in JavaScript methods, developers can write cleaner, more concise code, improving readability and efficiency. The key to mastering and implementing **array** and **string** methods lies in continual practice and exploration. There are many more built-in methods to be explored.

MAPS

Maps store 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. There are more examples available if you follow the link.

```
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:

- `set()` - adds or updates the key-value pairs in a map
- `has()` - checks whether 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

We've covered quite a lot of new concepts today. Now it's time to put them into practice to deepen your understanding. Let's dive in!

Instructions

To become more comfortable with the concepts covered in this task, read and run the accompanying examples files provided before doing the Compulsory Tasks.

Note: You will need to create an HTML file when needing to get input from a user. Please refer to previous learning material for a refresher if needed.

Compulsory Task 1

Follow these steps:

- Create a new JavaScript file in your task folder called **guestList.js**.
- Ask the user to input the names of people they would like to invite to a dinner party.
- Each name should be added to an array.
- When the user inputs the eleventh name, the program should return a message in the prompt box which reads: “You can only add a maximum of 10 people to your guest list. Would you like to replace someone on the list with the last person you entered? yes/no: ”
- If the user enters “yes”, the program should ask the user to, “Please enter the name of the person you would like to replace”. Note: The program should replace a person from the first 10 guests with the 11th person at the index position of the person they select to replace. The updated list with the replacement should then be displayed.
- If the user enters “no”, the program should only display the guest list consisting of the first 10 people that the user originally input.
- Ensure that the program handles all variants of yes/no responses from the user meaning YES, yes, NO, no, etc. should all be recognised.
- Display the guest-list that is output on a single line with each name separated by a comma.

Compulsory Task 2

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.