Introduction

For both front-end and back-end web development, JavaScript is a popular high-level, dynamic, and interpreted programming language. The array is one of the most significant data structures in JavaScript and is used to hold a group of values in a single variable. In this document I will examine, via examples, how to use arrays in JavaScript.

Research

Creating an Array

Creating an Array In JavaScript, you can either use the Array constructor or the square bracket syntax [] to build an array. Here are a few examples:

Using square bracket notation

```
let fruits = ['apple', 'banana', 'orange'];
```

Using Array constructor

```
let numbers = new Array(1, 2, 3, 4, 5);
```

Using the square bracket notation, I produce an array of fruits in the first example. In the second example, I make an array of numbers using the Array constructor.

Accessing Array Elements

The index of an array element can be used to access it. For the first element, the index is 0, and for the last element, it is the length of the array minus one. Here are a few examples:

let fruits = ['apple', 'banana', 'orange'];

console.log(fruits[0]); - output: apple

console.log(fruits[1]); - output: banana

console.log(fruits[2]); - output: orange

The values of the array's first three items are printed using console.log() in this example after I create an array of fruits.

Array Elements Updating

By putting a new value on an existing index, you can update the contents of an array. Here's an example:

let fruits = ['apple', 'banana', 'orange'];

fruits[1] = 'kiwi';

console.log(fruits); - output: ['apple', 'kiwi', 'orange']

In this example, I build a list of fruits and then change the second element from "banana" to "kiwi."

Array Elements Adding

The push() method adds the element to the end of the array when you add elements to an array. Here's an example:

let fruits = ['apple', 'banana', 'orange'];

fruits.push('kiwi');

console.log(fruits); - output: ['apple', 'banana', 'orange', 'kiwi']

In this example, I build an array of fruits and then add the word "kiwi" to the end of the array using the push() method.

Removing Array Elements

The pop() method, which eliminates the last element from the array, can be used to delete elements from an array. Here's an example:

let fruits = ['apple', 'banana', 'orange'];

fruits.pop();

console.log(fruits); - output: ['apple', 'banana']

In this example, I establish an array of fruits and then use the pop() method to get rid of the array's final element, "orange."

For Each Method

The forEach() method executes a provided function once for each array element.

```
let fruits = ['apple', 'banana', 'orange'];

fruits.forEach(function(fruit, index, fArray){

console.log(fruit); - output: shows all elements in array

console.log(index); - output: shows index of all elements in array (0, 1, 2 ...)

console.log(fArray); - output: shows the whole array

}
```

Map Method

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

```
let fruits = ['apple', 'banana', 'orange'];

const newFruits = fruits.map(fruit => {

return fuit

})

console.log(newFruits); - output: shows array newFruits which looks the same as fruits array


let fruits = ['apple', 'banana', 'orange'];

const newFruits = fruits.map(fruit => {

return 'Hello'

})

console.log(newFruits); - output: shows array newFruits which looks like [‘Hello’, ‘Hello’ ...]
```

Filter Method

The filter() method creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

```
let fruits = ['apple', 'banana', 'orange'];

const appFruits = []

For(let i = 0; i < fruits.length; i++){

If(fruits[i].includes(“app”))

{
```

```
AppFruits.push(fruits[i])

}

}

console.log(appFruits); - output: [“apple”]
```

Reduce Method

The reduce() method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

The first time that the callback is run there is no "return value of the previous calculation". If supplied, an initial value may be used in its place. Otherwise the array element at index 0 is used as the initial value and iteration starts from the next element (index 1 instead of index 0).

Perhaps the easiest-to-understand case for reduce() is to return the sum of all the elements in an array:

```
let fruits = [1, 2, 3];

const sum = fruits.reduce((total, fruit) => {

Return total + fruit

}, 0)

console.log(sum); - output: 6
```

Find Method

The find() method returns the first element in the provided array that satisfies the provided testing function.

```
let fruits = ['apple', 'banana', 'orange'];

const f = fruits.find(fruit => fruit === ‘apple’)

console.log(f); - output: apple
```

FindIndex Method

The findIndex() method returns the index of the first element in an array that satisfies the provided testing function. If no elements satisfy the testing function, -1 is returned.

```
let fruits = ['apple', 'banana', 'orange'];
```

const f = fruits.find(fruit => fruit === ‘apple’)

console.log(f); - output: 0

Conclusion

For the purpose of storing a number of values in a single variable, arrays are a crucial data structure in JavaScript. The Array constructor or the square bracket notation can be used to create arrays, and their index can be used to access individual elements. A variety of techniques can be used to update, add, and remove elements from an array. We can produce more effective and potent JavaScript code for web development by knowing how to use arrays.