Agenda

- Introduction to Arrays
- Array Properties
- Array Operations
- Inbuilt Array Functions
- Array Methods

An array is an object that can store multiple values at once. For example,

```
const words = ['hello', 'world', 'almabetter'];
```

Here, words is an array. The array is storing 3 values.

Create an Array

You can create an array using two ways:

1. Using an array literal

The easiest way to create an array is by using an array literal `[]` . For example,

```
const array1 = ["eat", "sleep"];
```

2. Using the new keyword

You can also create an array using JavaScript's `new` keyword.

```
const array2 = new Array("eat", "sleep");
```

In both of the above examples, we have created an array having two elements.

Note: It is recommended to use array literal to create an array.

Here are more examples of arrays:

```
// empty array
const myList = [ ];

// array of numbers
const numberArray = [ 2, 4, 6, 8];

// array of strings
const stringArray = [ 'eat', 'work', 'sleep'];

// array with mixed data types
const newData = ['work', 'exercise', 1, true];
```

You can also store arrays, functions and other objects inside an array. For example,

```
const newData = [
    {'task1': 'exercise'},
    [1, 2 ,3],
    function hello() { console.log('hello')}
```

```
];
```

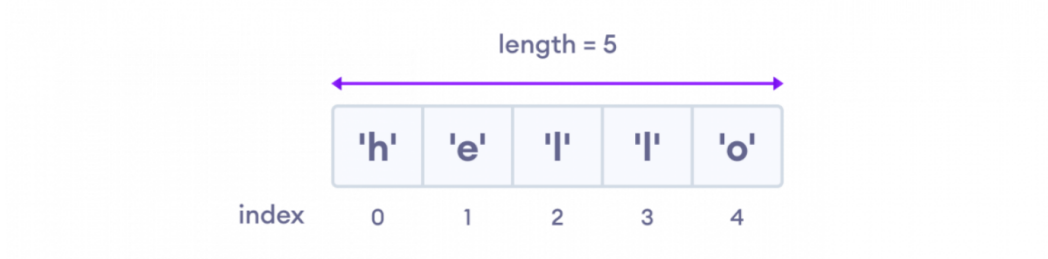
Access Elements of an Array

You can access elements of an array using indices **(0, 1, 2 ...)**. For example,

```
const myArray = ['h', 'e', 'l', 'l', 'o'];

// first element
console.log(myArray[0]);  // "h"

// second element
console.log(myArray[1]); // "e"
```



> Array indexing in JavaScript Note : Array's index starts with 0, not 1.

Add an Element to an Array

You can use the built-in method `push()` and `unshift()` to add elements to an array.

The `push()` method adds an element at the end of the array. For example,

```
let dailyActivities = ['eat', 'sleep'];

// add an element at the end
dailyActivities.push('exercise');

console.log(dailyActivities); //  ['eat', 'sleep', 'exercise']
```

The `unshift()` method adds an element at the beginning of the array. For example,

```
let dailyActivities = ['eat', 'sleep'];

//add an element at the start
dailyActivities.unshift('work');

console.log(dailyActivities); // ['work', 'eat', 'sleep']
```

Change the Elements of an Array

You can also add elements or change the elements by accessing the index value.

```
let dailyActivities = [ 'eat', 'sleep'];

// this will add the new element 'exercise' at the 2 index
dailyActivities[2] = 'exercise';

console.log(dailyActivities); // ['eat', 'sleep', 'exercise']
```

Suppose an array has two elements. If you try to add an element at index 3 (fourth element), the third element will be undefined. For example,

```
let dailyActivities = [ 'eat', 'sleep'];

// this will add the new element 'exercise' at the 3 index
dailyActivities[3] = 'exercise';

console.log(dailyActivities); // ["eat", "sleep", undefined, "exercise"]
```

Basically, if you try to add elements to high indices, the indices in between will have an undefined value.

Remove an Element from an Array

You can use the `pop()` method to remove the last element from an array. The `pop()` method also returns the returned value. For example,

```
let dailyActivities = ['work', 'eat', 'sleep', 'exercise'];

// remove the last element
dailyActivities.pop();
console.log(dailyActivities); // ['work', 'eat', 'sleep']

// remove the last element from ['work', 'eat', 'sleep']
const removedElement = dailyActivities.pop();

//get removed element
console.log(removedElement); // 'sleep'
console.log(dailyActivities);  // ['work', 'eat']
```

If you need to remove the first element, you can use the `shift()` method. The `shift()` method removes the first element and also returns th removed element. For example,

```
let dailyActivities = ['work', 'eat', 'sleep'];

// remove the first element
dailyActivities.shift();

console.log(dailyActivities); // ['eat', 'sleep']
```

Array length

You can find the length of an element (the number of elements in an array) using the `length` property. For example,

```
const dailyActivities = [ 'eat', 'sleep'];

// this gives the total number of elements in an array
```

```
console.log(dailyActivities.length); // 2
```

Array Operations

In JavaScript, there are various array methods available that make it easier to perform useful calculations.

Some of the commonly used JavaScript array methods are:

Method	Description
concat()	joins two or more arrays and returns a result
indexOf()	searches an element of an array and returns its position
filter()	creates a new array filled with elements that pass a test provided by a function
find()	returns the first value of an array element that passes a test
findIndex()	returns the first index of an array element that passes a test
forEach()	calls a function for each element
includes()	checks if an array contains a specified element
map()	creates a new array from calling a function once for every array element.
push()	aads a new element to the end of an array and returns the new length of an array
unshift()	adds a new element to the beginning of an array and returns the new length of an array
pop()	removes the last element of an array and returns the removed element
shift()	removes the first element of an array and returns the removed element
sort()	sorts the elements alphabetically in strings and in ascending order
slice()	selects the part of an array and returns the new array
splice()	removes or replaces existing elements and/or adds new elements

Example: JavaScript Array Methods

```
let dailyActivities = ['sleep', 'work', 'exercise']
let newRoutine = ['eat'];
let ages = [4, 9, 16, 25, 36, 49];

// sorting elements in the alphabetical order
dailyActivities.sort();
console.log(dailyActivities); // ['exercise', 'sleep', 'work']

//finding the index position of string
const position = dailyActivities.indexOf('work');
console.log(position); // 2

// slicing the array elements
const newDailyActivities = dailyActivities.slice(1);
console.log(newDailyActivities); // [ 'sleep', 'work']

// concatenating two arrays
const routine = dailyActivities.concat(newRoutine);
console.log(routine); // ["exercise", "sleep", "work", "eat"]

// filter an array
const adults = ages.filter(checkAdult);
function checkAdult(age) {
  return age >= 18;
}
console.log(adults); // [25, 36, 49]
```

```
//map a function to an array
const ages_sqrt = ages.map(Math.sqrt)
console.log(ages_sqrt); // [2, 3, 4, 5, 6, 7]
```

> Note: If the element is not in an array, indexOf() gives -1.

Working of JavaScript Arrays

In JavaScript, an array is an object. And, the indices of arrays are objects keys.

Since arrays are objects, the array elements are stored by reference. Hence, when an array value is copied, any change in the copied array will als reflect in the original array. For example,

```
let arr = ['h', 'e'];
let arr1 = arr;
arr1.push('l');

console.log(arr); // ["h", "e", "l"]
console.log(arr1); // ["h", "e", "l"]
```

You can also store values by passing a named key in an array. For example,

```
let arr = ['h', 'e'];
arr.name = 'John';

console.log(arr); // ["h", "e"]
console.log(arr.name); // "John"
console.log(arr['name']); // "John"
```

Array indexing in JavaScript

However, it is not recommended to store values by passing arbitrary names in an array.

Hence in JavaScript, you should use an array if values are in an ordered collection. Otherwise, it's better to use object with `{ }` .

A multidimensional array is an array that contains another array. For example,

```
// multidimensional array
const data = [[1, 2, 3], [1, 3, 4], [4, 5, 6]];
```

Create a Multidimensional Array

Here is how you can create multidimensional arrays in JavaScript.

Example 1

```
let studentsData = [['Jack', 24], ['Sara', 23], ['Peter', 24]];
```

Example 2

```
let student1 = ['Jack', 24];
let student2 = ['Sara', 23];
let student3 = ['Peter', 24];

// multidimensional array
let studentsData = [student1, student2, student3];
```

Here, both example 1 and example 2 creates a multidimensional array with the same data.

Access Elements of an Array

You can access the elements of a multidimensional array using indices **(0, 1, 2 ...)**. For example

```
let x = [
['Jack', 24],
['Sara', 23],
['Peter', 24]
];

// access the first item
console.log(x[0]); // ["Jack", 24]

// access the first item of the first inner array
console.log(x[0][0]); // Jack

// access the second item of the third inner array
console.log(x[2][1]); // 24
```

You can think of a multidimensional array (in this case, x), as a table with 3 rows and 2 columns.

	Column 1	Column 2
Row 1	Jack x[0][0]	24 x[0][1]
Row 2	Sara x[1][0]	23 x[1][1]
Row 3	Peter x[2][0]	24 x[2][1]

Accessing multidimensional array elements.

Add an Element to a Multidimensional Array

You can use the Array's push() method or an indexing notation to add elements to a multidimensional array.

Adding Element to the Outer Array

```
let studentsData = [['Jack', 24], ['Sara', 23],];
studentsData.push(['Peter', 24]);
```

```
console.log(studentsData); //[["Jack", 24], ["Sara", 23], ["Peter", 24]
```

Adding Element to the Inner Array

```
// using index notation
let studentsData = [['Jack', 24], ['Sara', 23],];
studentsData[1][2] = 'hello';

console.log(studentsData); // [["Jack", 24], ["Sara", 23, "hello"]]
```

```
// using push()
let studentsData = [['Jack', 24], ['Sara', 23],];
studentsData[1].push('hello');

console.log(studentsData); // [["Jack", 24], ["Sara", 23, "hello"]]
```

You can also use the Array's splice() method to add an element at a specified index. For example,

```
let studentsData = [['Jack', 24], ['Sara', 23],];

// adding element at 1 index
studentsData.splice(1, 0, ['Peter', 24]);

console.log(studentsData); // [["Jack", 24], ["Peter", 24], ["Sara", 23]]
```

Remove an Element from a Multidimensional Array

You can use the Array's pop() method to remove the element from a multidimensional array. For example,

Remove Element from Outer Array

```
// remove the array element from outer array
let studentsData = [['Jack', 24], ['Sara', 23],];
studentsData.pop();

console.log(studentsData); // [["Jack", 24]]
```

Remove Element from Inner Array

```
// remove the element from the inner array
let studentsData = [['Jack', 24], ['Sara', 23]];
studentsData[1].pop();

console.log(studentsData); // [["Jack", 24], ["Sara"]]
```

You can also use the `splice()` method to remove an element at a specified index. For example,

```
let studentsData = [['Jack', 24], ['Sara', 23],];

// removing 1 index array item
studentsData.splice(1,1);
console.log(studentsData); // [["Jack", 24]]
```

Iterating over Multidimensional Array

You can iterate over a multidimensional array using the Array's forEach() method to iterate over the multidimensional array. For example,

```
let studentsData = [['Jack', 24], ['Sara', 23],];

// iterating over the studentsData
studentsData.forEach((student) => {
    student.forEach((data) => {
        console.log(data);
    });
});
```

Output

```
Jack
24
Sara
23
```

The first `forEach()` method is used to iterate over the outer array elements and the second `forEach()` is used to iterate over the inner arra
elements.

You can also use the `for...of` loop to iterate over the multidimensional array. For example,

```
let studentsData = [['Jack', 24], ['Sara', 23],];

for (let i of studentsData) {
  for (let j of i) {
    console.log(j);
  }
}
```

You can also use the for loop to iterate over a multidimensional array. For example,

```
let studentsData = [['Jack', 24], ['Sara', 23],];

// looping outer array elements
for(let i = 0; i < studentsData.length; i++){

    // get the length of the inner array elements
    let innerArrayLength = studentsData[i].length;
```

```
    // looping inner array elements
    for(let j = 0; j < innerArrayLength; j++) {
        console.log(studentsData[i][j]);
    }
}
```

Interview Questions

> How can you double elements of an array using reduce? Please note that you cannot create additional variables.

```
const arrayOfNumbers = [1, 2, 3, 4];
arrayOfNumbers.reduce((accumulator, currentValue, index, array) => array[index] = array[index] * 2);
```

> Can you describe the main difference between a `.forEach` loop and a `.map()` loop and why you would pick one versus the other? **`forEach`**

- Iterates through the elements in an array.
- Executes a callback for each element.
- Does not return a value.

```
const a = [1, 2, 3];
const doubled = a.forEach((num, index) => {
  // Do something with num and/or index.
});

// doubled = undefined
```

`map`

- Iterates through the elements in an array.
- "Maps" each element to a new element by calling the function on each element, creating a new array as a result.

```
const a = [1, 2, 3];
const doubled = a.map(num => {
  return num * 2;
});

// doubled = [2, 4, 6]
```

The main difference between `.forEach` and `.map()` is that `.map()` returns a new array. If you need the result, but do not wish to mutate th original array, `.map()` is the clear choice. If you simply need to iterate over an array, `forEach` is a fine choice.