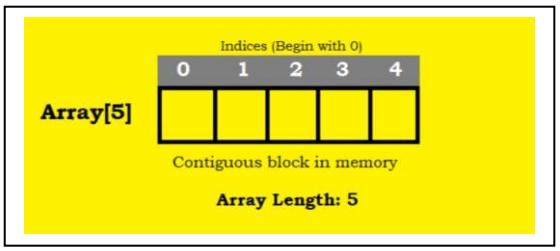
ARRAY

In JavaScript, an array is a data structure that allows you to store multiple values in a single variable. These values can be of any data type, including numbers, strings, objects, or even other arrays, making arrays versatile for organizing and manipulating data.



Arrays in JavaScript are zero-indexed, meaning the first element is accessed with the index 0, the second element with index 1, and so on. You can access and modify elements in an array using their index.

ARRAYS CAN BE DECLARED USING THE ARRAY CONSTRUCTOR OR THE [] NOTATION

Syntax: Use the following syntax to create an Array Object
e.g.-1
var fruits = new Array ("apple", "orange", "mango");
e.g.-2

```
<script>
 var fruits = new Array ("apple", "orange", "mango" );
 console.log(fruits)
 console.log(fruits[0])
 console.log(fruits[1])
 console.log(fruits[2])
</script>
```

Array Literals : You can create array by simply assigning values as follows:
e.g.
var fruits = ["apple", "orange", "mango"]

Array Properties

Constructor

The constructor of an array in JavaScript refers to the function used to create instances of arrays. In JavaScript, arrays are objects, and their constructor function is Array(). When you create an array using the array literal notation [] or the new Array() constructor, you're essentially invoking the Array() constructor function behind the scenes.
e.g.

```
<html>
 <body>
  <script>
   var arr = new Array(10, 20, 30);
   console.log(arr.constructor === Array);
  </script>
 </body>
</html>
```

Length

The length property of an array in JavaScript returns the number of elements in that array. It represents the highest numeric index plus one. It returns an ***unsigned, 32-bit integer*** (-2147483648 to +2147483648) that specifies the number of elements in an array.

e.g.

```
<html>
 <body>
  <script>
   var arr = new Array(10, 20, 30);
   document.write("Length is:" + arr.length);
  </script>
 </body>
</html>
```

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];

e.g.

```
<html>
 <body>
  <script>
     const newData = ['work', 'exercise', 1, true];
     document.writeln(typeof(newData[0]))
     document.writeln(typeof(newData[1]))
     document.writeln(typeof(newData[2]))
     document.writeln(typeof(newData[3]))
  </script>
 </body>
</html>
```

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

```
const newData = [
        [1, 2 ,3],
        function hello() { console.log('hello')}
];
        e.g.
```

e.g.

```
<html>
 <body>
  <script>
    const newData = [
        [1, 2 ,3],
        function hello() { console.log('hello')}
     ];

     console.log(typeof newData[0])
     console.log(typeof newData[1])
```

```
  </script>
 </body>
</html>
```

Access Elements of an Array

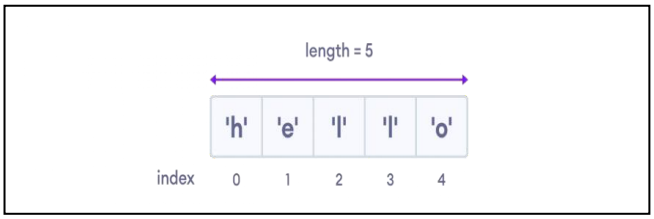
Accessing elements of an array in JavaScript refers to retrieving individual values stored within the array. Elements in an array are indexed starting from 0, with each element associated with a unique index.

e.g.

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

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

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



Note:

1. Array's index starts with 0.
2. Index number is also called subscript.
3. Smallest index is called lower bound and largest subscript is called upper bound.

e.g.

```
<html>
 <body>
  <script>
   let list = new Array(5,3,7,1,5)
   let sum = 0 ;
   for ( i = 0 ; i< list.length ; i++ ){
    sum = sum + list[i];
   }
   const avg = sum / list.length;
   document.write("average : " + avg )
  </script>
 </body>
</html>
```

Add an Element to an Array

Adding elements to an array in JavaScript is a common operation, and there are several methods available to achieve this.

push()

The push() method in JavaScript is used to add one or more elements to the end of an array. It modifies the original array by appending the new elements and returns the updated length of the array.

Syntax:

array.push(element1, element2, ..., elementN);

e.g.-1
```
<script>
  let dailyActivities = ['eat', 'sleep'];
    dailyActivities.push('exercise');
    console.log(dailyActivities);
</script>
```

e.g.-2
```
<script>
  let dailyActivities = ['eat', 'sleep'];
  x = dailyActivities.push('exercise');
  console.log(x)
  console.log(dailyActivities);
</script>
```

unshift()

The unshift() method in JavaScript is used to add one or more elements to the beginning of an array. It modifies the original array by shifting existing elements to higher indices to accommodate the new elements and returns the new length of the array.

Syntax:

```
array.unshift(element1, element2, ..., elementN);
```

e.g.
```
<script>
  let dailyActivities = ['eat', 'sleep'];
    dailyActivities.unshift('exercise');
    console.log(dailyActivities);
</script>
```

Change the Elements of an Array

To change the elements of an array in JavaScript, you can directly assign new values to specific indices of the array or use array methods to modify the elements.

e.g.
```
<script>
  let dailyActivities = [ 'eat', 'sleep'];
  dailyActivities[1] = 'exercise';
  console.log(dailyActivities);
</script>
```

Note:

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

e.g.
```
   <script>
      let dailyActivities = ['eat', 'sleep'];
      dailyActivities[3] = 'exercise';
      document.write(dailyActivities)
      console.log(dailyActivities);
   </script>
```

Remove an Element from an Array

Removing elements from an array in JavaScript can be done using various methods, depending on the specific requirement and the desired effect on the array.

pop()

The pop() method in JavaScript is used to remove the last element from an array and returns that removed element. This method modifies the original array by removing the last element and reducing its length by one.

Syntax:

```
array.pop();
```

e.g.

```
<script>
  let dailyActivities = ['work', 'eat', 'sleep']
  dailyActivities.pop();
  console.log(dailyActivities);
</script>
```

shift()

The shift() method in JavaScript is used to remove the first element from an array and returns that removed element. This method modifies the original array by removing the first element and shifting all subsequent elements to a lower index.

Syntax:

```
array.shift();
```

e.g

```
<script>
  let dailyActivities = ['work', 'eat', 'sleep'];
  let x = dailyActivities.shift();
  console.log(dailyActivities);
  document.write("deleted element : " + x)
</script>
```

Sort()

The sort() method in JavaScript is used to sort the elements of an array in place and returns the sorted array. By default, the elements are sorted in ascending order according to their Unicode code
points.

e.g.

```
 <script>
     let dailyActivities = ['sleep', 'work', 'exercise']
     dailyActivities.sort();
     console.log(dailyActivities);
 </script>
```

indexOf()

The indexOf() method in JavaScript is used to find the first occurrence of a specified value within an array and returns the index at which it is located. If the value is not found, -1 is returned.

Syntax

array.indexOf(searchElement, startIndex);

e.g.-1

```
<script>
  let dailyActivities = ['sleep', 'work', 'exercise']
  const position = dailyActivities.indexOf('work');
  console.log(position);
</script>
```

e.g.-2

```
let fruits = ["apple", "banana", "orange", "banana"];
let bananaIndex = fruits.indexOf("banana", 2);
console.log(bananaIndex); // Output: 3
```

slice()

The slice() method in JavaScript is used to extract a section of an array and returns a new array containing the extracted elements. The original array is not modified.

Syntax

array.slice(start, end);

e.g.

```
<script>
  let fruits = ["apple", "banana", "orange", "kiwi", "mango"];
  let slicedFruits = fruits.slice(1, 4);
  console.log(slicedFruits); // Output: ["banana", "orange", "kiwi"]
</script>
```

concat()

The concat() method in JavaScript is used to merge two or more arrays, or values, into a new array. It does not modify the existing arrays but returns a new array containing the elements of the original arrays along with any additional elements passed as arguments. Here's an explanation of how the concat() method works:

Syntax:

e.g.

```
   <script>
      let dailyActivities = ['sleep', 'work', 'exercise']
      let newRoutine = ["plan1", "plan2"]
      const routine = dailyActivities.concat(newRoutine);
      console.log(routine);
   </script>
```

WORKING OF JAVASCRIPT ARRAYS

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 also reflect in the original array.

e.g.

```
<html>
<body>
   <script>
      let arr = ['h', 'e'];
      let arr1 = arr;
      arr1.push('l');
      console.log(arr); // ["h", "e", "l"]
      console.log(arr1); // ["h", "e", "l"]
      arr[1] ='p';
      console.log(arr); // ["h", "p", "l"]
      console.log(arr1); // ["h", "p", "l"]
   </script>
</body>
</html>
```

You can also store values by passing a named key in an array

e.g.

```
<html>
<body>
   <script>
     let arr = ['h', 'e'];
      arr.name = 'amit';
      arr.phone_no = 9999988888;
      console.log(arr); // ["h", "e"]
      console.log(arr.name); // "amit"
      console.log(arr.phone_no); // "amit"
      console.log(arr['name']); // "amit"
      console.log(arr['phone_no']); // "amit"
   </script>
</body>
</html>
```

Akhilesh Kumar Gupta
(Technical Training specialist)
TechCarrel LLP