JavaScript - Array

The JavaScript **Array** object lets you store multiple values in a single variable. An array is used to store a sequential collection of multiple elements of same or different data types. In JavaScript, arrays are dynamic, so you don't need to specify the length of the array while defining the array. The size of a JavaScript array may decrease or increase after its creation.

Syntax

Use the following syntax to create an array object in JavaScript −

```
const arr = new Array(val1, val2, val3, ..., valN)
```

Parameters

val1, val2, val3, ..., valN − It takes multiple values as an argument to initialize an array with them.

Learn **JavaScript** in-depth with real-world projects through our **JavaScript certification course**. Enroll and become a certified expert to boost your career.

Return value

It returns an array object.

Note − When you pass the single numeric argument to the Array() constructor, it defines the array of argument length containing the undefined values. The maximum length allowed for an array is 4,294,967,295.

You can add multiple comma separated elements inside square brackets to create an

array using the array literal −

```
const fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

```
fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element
```

JavaScript Array Reference

In JavaScript, the Array object enables storing collection of multiple elements under a single variable name. It provides various methods and properties to perform operations on an array. Here, we have listed the properties and methods of the Array class.

JavaScript Array Properties

Here is a list of the properties of the Array object along with their description −

Sr.No.	Name & Description
1	**constructor** Returns a reference to the array function that created the object.
2	**length** Reflects the number of elements in an array.

JavaScript Array Methods

Here is a list of the methods of the Array object along with their description −

Array Static methods

These methods are invoked using the Array class itself −

Sr.No.	Name & Description

1	**from()** Creates a shallow copy of the array.
2	**isArray()** Returns boolean values based on the argument is an array.
3	**Of()** Creates an array from multiple arguments.

Array instance methods

These methods are invoked using the instance of the Array class –

Sr.No.	Name & Description
1	**at()** To get element from the particular index.
2	**concat()** Returns a new array comprised of this array joined with another array (s) and/or value(s).
3	**copyWithin()** To Copy part of the array into the same array at different locations.
4	**entries()** To get each entry of the array.
5	**every()** Returns true if every element in this array satisfies the provided testing function.
6	**fill()** To fill the array with static values.
7	**filter()** Creates a new array with all of the elements of this array for which the provided filtering function returns true.
8	**find()** To find an element satisfying the condition.
9	**findIndex()** To find an index of the element satisfying the condition.
10	**findLast()** To find an element satisfying the condition from the last.

11	**findLastIndex()** To find an index of the element satisfying the condition from the last.
12	**flat()** To flatten the array.
13	**flatMap()** To get a new array after flattening the array.
14	**forEach()** Calls a function for each element in the array.
15	**Includes()** Returns a boolean value if the array contains the specific element.
16	**indexOf()** Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
17	**join()** Joins all elements of an array into a string.
18	**Keys()** Returns an array iterator containing the key for each array element.
19	**lastIndexOf()** Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
20	**map()** Creates a new array with the results of calling a provided function on every element in this array.
21	**pop()** Removes the last element from an array and returns that element.
22	**push()** Adds one or more elements to the end of an array and returns the new length of the array.
23	**reduce()** Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
24	**reduceRight()** Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

25	**reverse()** Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
26	**shift()** Removes the first element from an array and returns that element.
27	**slice()** Extracts a section of an array and returns a new array.
28	**some()** Returns true if at least one element in this array satisfies the provided testing function.
29	**toSorted()** Sorts the elements of an array in a specific order.
30	**sort()** Sorts the elements of an array.
31	**splice()** Adds and/or removes elements from an array.
32	**toLocaleString()** To convert array elements into the string.
33	**toReversed()** Returns a reverse of the array.
34	**toSpliced()** This method returns a new array with some elements removed and/or replaced at a given index.
35	**toString()** Returns a string representing the array and its elements.
36	**unshift()** Adds one or more elements to the front of an array and returns the new length of the array.
37	**values()** To get an iterator containing values of each array index.
38	**with()** This method returns a new array with the element at the given index replaced with the given value.

Basic Examples of JavaScript Array Object

In the following examples, we have demonstrated the usage of basic methods and properties of JavaScript Array Object –

Example (Creating JavaScript Array Object)

In the example below, the array 'strs' is initialized with the string values passed as an Array() constructor's argument.

The 'cars' array contains 20 undefined elements. If you pass multiple numeric values, it defines the array containing those elements but needs to be careful with a single numeric argument to the array() constructor.

Open Compiler

```
<html>
<head>
   <title> JavaScript - Array() constructor </title>
</head>
<body>
   <p id = "demo"> </p>
   <script>
      const output = document.getElementById("demo");

      let strs = new Array("Hello", "World!", "Tutorials Point");
      output.innerHTML += "strs ==> " + strs + "<br>";

      let cars = new Array(20);
      output.innerHTML += "cars ==> " + cars + "<br>";
   </script>
</body>
</html>
```

Output

Execute the above program to see the desired output.

```
strs ==> Hello,World!,Tutorials Point
cars ==> ,,,,,,,,,,,,,,,,,,,
```

Example (Creating Arrays Using Array Literal)

In the example below, we have created different arrays. The arr1 array contains the numbers, the arr2 array contains the strings, and the arr3 array contains the boolean values.

Open Compiler

```
<html>
<head>
   <title> JavaScript - Array literals </title>
</head>
<body>
   <p id = "output"> </p>
   <script>
      const output = document.getElementById("output");

      const arr1 = [10, 40, 50, 60, 80, 90];  // Array of numbers
      const arr2 = ["Hello", "Hi", "How", "are", "you?"]; // Array of strings
      const arr3 = [true, false, true, true]; // Array of booleans

      output.innerHTML += "arr1 ==>  " + arr1 + "<br>";
      output.innerHTML += "arr2 ==>  " + arr2 + "<br>";
      output.innerHTML += "arr3 ==>  " + arr3 + "<br>";
   </script>
</body>
</html>
```

Output

Execute the above program to see the desired output.

```
arr1 ==> 10,40,50,60,80,90
arr2 ==> Hello,Hi,How,are,you?
arr3 ==> true,false,true,true
```

Accessing JavaScript Array Elements

The array index starts from 0. So, you can access the array element using its index.

```
let number = arr[index]
```

In the above syntax, 'arr' is an array, and 'index' is a number from where we need to access the array element.

Example

In the example below, we have created the array of numbers and accessed the elements from the 0th and 2nd index of the array. The element at the 0th index is 1, and the element at the 2nd index is 6.

Open Compiler

```
<html>
<head>
   <title> JavaScript - Accessing array elements </title>
</head>
<body>
   <p id = "output"> </p>
   <script>
      const nums = [1, 5, 6, 8, 90];
      document.getElementById("output").innerHTML =
      "Element at 0th index is : " + nums[0] + "<br>" +
      "Element at 2nd index is : " + nums[2];
   </script>
</body>
</html>
```

Output

```
Element at 0th index is : 1
Element at 2nd index is : 6
```

JavaScript Array length

The 'length' property of the array is used to find the length of the array.

```
let len = arr.length;
```

Example

In the example below, the 'length' property returns 5, as array contains 5 elements.

Open Compiler

```
<html>
<head>
   <title> JavaScript - Array length </title>
</head>
<body>
   <p id = "output"> </p>
   <script>
      const nums = [1, 5, 6, 8, 90];
      document.getElementById("output").innerHTML =
      "Array length is : " + nums.length;
   </script>
</body>
</html>
```

Output

```
Array length is : 5
```

Adding a new element to the array

You can use the push() method to insert the element at the end of the array. Another solution is that you can insert the array at the index equal to the array length.

```
arr.push(ele)
OR
arr[arr.length] = ele;
```

In the above syntax, 'ele' is a new element to insert into the array. Here, if the array length is N, the array contains elements from 0 to N – 1 index. So, we can insert the new element at the Nth index.

Example

In the example below, we insert 6 to the array using the push() method. Also, we used the 'length' property to insert the element at the end.

Open Compiler

```
<html>
<body>
   <p id = "output"> </p>
   <script>
      const output = document.getElementById("output");
      const nums = [1, 2, 3, 4, 5];
      nums.push(6); // Inserting 6 at the end
      output.innerHTML += "Updated array is : " + nums + "<br>";
      nums[nums.length] = 7; // Inserting 7
      output.innerHTML += "Updated array is : " + nums + "<br>"
   </script>
</body>
</html>
```

Output

As we can see in the output, provided element has been added.

```
Updated array is : 1,2,3,4,5,6
Updated array is : 1,2,3,4,5,6,7
```

Updating JavaScript Array Elements

To update any array element, you can access the array index and change its value.

```
arr[index] = ele;
```

In the above syntax, 'index' is an index where we need to update a value with the 'ele' value.

Example

In the example below, we update the element at the first index in the array.

Open Compiler

```
<html>
<body>
   <p id = "output"> </p>
   <script>
```

```
      const nums = [1, 2, 3, 4, 5];
      nums[0] = 100; // Updating first element
      document.getElementById("output").innerHTML =
      "Updated array is : " + nums;
   </script>
</body>
</html>
```

Output

Updated array is : 100,2,3,4,5

Traversing a JavaScript Array

You can use the loop to traverse through each array element. However, some built-in methods exist to traverse the array, which we will see in later chapters.

```
for (let p = 0; p < nums.length; p++) {
   // Access array using the nums[p]
}
```

Example

In the below code, the array contains 5 numbers. We used the for loop to traverse the array and print each element.

However, while and do-while loops can also be used to traverse the array.

Open Compiler

```
<html>
<body>
   <p id = "demo"> </p>
   <script>
      const output = document.getElementById("demo");
      const nums = [1, 2, 3, 4, 5];
      for (let p = 0; p < nums.length; p++) {
         output.innerHTML += "nums[" + p + "] ==> " + nums[p] + "<br>";
      }
   </script>
</body>
```

```
</html>
```

Output

```
nums[0] ==> 1
nums[1] ==> 2
nums[2] ==> 3
nums[3] ==> 4
nums[4] ==> 5
```