# Array Built-In Properties & Methods

## Array Properties

### length

Array includes "length" property which returns number of elements in the array.

Syntax

```
array.length
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
console.log(fruits.length);
```

Example

```
var arr = [20,30,"Hello"];
arr.length = 5;
console.log(arr);
```

## Array Methods

### 1) concat()

- Returns new array by combining values of an array that is specified as parameter with existing array values.
- This method does not change the existing arrays, but returns a new array, containing the values of the joined arrays.

Syntax

```
array1.concat(array2, array3, ..., arrayX)
```

Example

```
var pets = ["Cat", "Dog", "Parrot"];
var wilds = ["Tiger", "Wolf", "Zebra"];
var animals = pets.concat(wilds);
console.log(animals);
```

Example

```
var hege = ["Cecilie", "Lone"];
var stale = ["Emil", "Tobias", "Linus"];
var children = hege.concat(stale);
console.log(children);
```

**2) copyWithin()**

- This method copies array elements to another position in the array, overwriting the existing values.
- This method will never add more items to the array.

Syntax

```
array.copyWithin(target, start, end)
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var bac = fruits.copyWithin(2, 0);
console.log(bac);
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango", "Kiwi", "Papaya"];
var bac = fruits.copyWithin(2, 0, 2);
console.log(bac)
```

**3) entries()**

- This method returns an Array Iterator object with key/value pairs.
- This method does not change the original array.

Syntax

```
array.entries()
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var f = fruits.entries();
for (x of f) {
  console.log(x);
}
```

**4) every()**

- This method checks if all elements in an array pass a test.
- This method does not change the original array.
- The every() method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.
- Here the test that is passed by the function is performed by another function.

Syntax

```
array.every(function(currentValue, index, arr));
```

Example

```
function isBelowThreshold(currentValue) {
  return currentValue < 40;
}
var array1 = [1, 30, 39, 29, 10, 13];
console.log(array1.every(isBelowThreshold));
```

Example

```
var ages = [32, 33, 16, 40];
function checkAdult(age) {
  return age >= 18;
}
function myFunction() {
  console.log(ages.every(checkAdult));
}
myFunction();
```

**5) fill()**

- The fill() method fills (modifies) all the elements of an array from a start index (default zero) to an end index (default array length) with a static value.
- It returns the modified array.

Syntax

```
array.fill(value, start, end)
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var bac = fruits.fill("Kiwi");
console.log(bac);
```

**6) filter()**

- The filter() method creates a new array with all elements that pass the test implemented by the provided function.
- This method does not execute the function for array elements without values.
- This method does not change the original array.

Syntax

```
array.filter(function(currentValue, index, arr))
```

Example

```
var ages = [32, 33, 16, 40];
function checkAdult(age) {
  return age >= 18;
}
function myFunction() {
  console.log(ages.filter(checkAdult))
}
myFunction()
```

**7) find()**

- The find() method returns the value of the first element in the array that satisfies the provided testing function.
- This method does not execute the function for empty arrays.

Syntax

```
array.find(function(currentValue, index, arr))
```

Example

```
var array1 = [5, 12, 8, 130, 44];
var found = array1.find(function(element) {
  return element > 10;
});
console.log(found);
```

Example

```
var ages = [3, 10, 18, 20];
function checkAdult(age) {
  return age >= 18;
}
function myFunction() {
  console.log(ages.find(checkAdult));
}
myFunction();
```

**8) findIndex()**

- The findIndex() method returns the index of the first element in the array that satisfies the provided testing function.
- It returns -1, indicating that no element passed the test.

Syntax

```
array.findIndex(function(currentValue, index, arr), thisValue)
```

Example

```
var array1 = [5, 12, 8, 130, 44];
function isLargeNumber(element) {
  return element > 13;
}
console.log(array1.findIndex(isLargeNumber));
```

**9) forEach()**

- The forEach() method executes a provided function once for each array element.
- This function is not executed for array elements without values.

Syntax

```
array.forEach(function(currentValue, index, arr), thisValue)
```

Example

```
var array1 = ['a', 'b', 'c'];
array1.forEach(function(element) {
  console.log(element);
});
```

**10) from()**

The Array.from() method returns an Array object from any object with a length property or an iterable object.

Syntax

```
Array.from(object, mapFunction, thisValue)
```

Example

```
var myArr = Array.from("ABCDEFG");
console.log(myArr);
```

**11) includes()**

- The includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.
- The includes() method is case sensitive.

Syntax

```
array.includes(element, start)
```

Example

```
var array1 = [1, 2, 3];
console.log(array1.includes(2));
```

Example

```
var pets = ['cat', 'dog', 'bat'];
console.log(pets.includes('cat'));
```

**12) indexOf()**

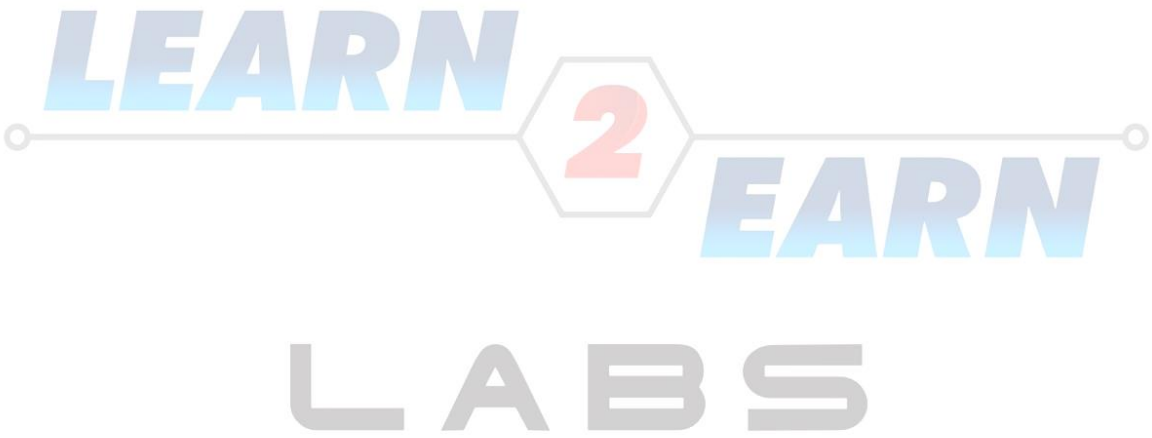
- The indexOf() method searches the array for the specified item, and returns its position.
- The search will start at the specified position, or at the beginning if no start position is specified, and end the search at the end of the array.
- Returns -1 if the item is not found.

Syntax

```
array.indexOf(item, start)
```

Example

```
var beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];
console.log(beasts.indexOf('bison'));
```



**13) isArray()**

- The isArray() method determines whether an object is an array.
- This function returns true if the object is an array, and false if not.

Syntax

```
Array.isArray(obj)
```

Example

```
function abc() {
  var fruits = ["Banana", "Orange", "Apple", "Mango"];
  console.log(Array.isArray(fruits))
}
abc()
```

**14) join()**

The join() method creates and returns a new string by concatenating all of the elements in an array (or an array-like object), separated by commas or a specified separator string.

Syntax

```
array.join(separator)
```

Example

```
var elements = ['Fire', 'Air', 'Water'];
console.log(elements.join());
console.log(elements.join(''));
console.log(elements.join('-'));
```

**15) keys()**

- The keys() method returns a new Array Iterator object that contains the keys for each index in the array.
- This method does not change the original array.

Syntax

```
array.keys()
```

Example

```
var array1 = ['a', 'b', 'c'];
var iterator = array1.keys();
  for (let key of iterator) {
  console.log(key);
}
```

**16) lastIndexOf()**

The lastIndexOf() method returns the last index at which a given element can be found in the array, or -1 if it is not present.

Syntax

```
array.lastIndexOf(item, start)
```

Example

```
var animals = ['Dodo', 'Tiger', 'Penguin', 'Dodo'];
console.log(animals.lastIndexOf('Dodo'));
console.log(animals.lastIndexOf('Tiger'));
```

**17) map()**

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

Syntax

```
array.map(function(currentValue, index, arr), thisValue)
```

Example

```
var abc = [10,"Hello",”boy”];
abc.map(function(currentValue,index,arr){
  console.log(currentValue,index)
})
```

**18) pop()**

- The pop() method removes the last element from an array and returns that element.
- This method changes the length of the array.

Syntax

```
array.pop()
```

Example

```
var plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];
console.log(plants.pop());
console.log(plants);
```

**19) push()**

- The push() method adds one or more elements to the end of an array and returns the new length of the array.
- This method changes the length of the array.
- The new item(s) will be added at the end of the array.

Syntax

```
array.push(item1, item2, ..., itemX)
```

Example

```
var animals = ['pigs', 'goats', 'sheep'];
console.log(animals.push('cows'));
```

**20) reduce()**

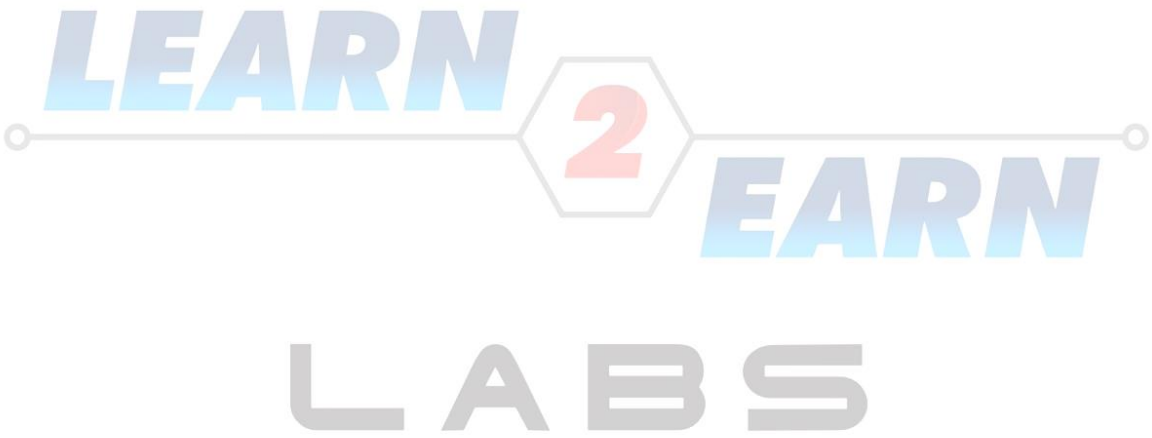
- The reduce() method executes a reducer function (that you provide) on each element of the array, resulting in a single output value.
- The reduce() method reduces the array to a single value.

Syntax

```
array.reduce(function(total, currentValue, currentIndex, arr), initialValue)
```

Example

```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;
console.log(array1.reduce(reducer)); // expected output: 10
```



**21) reduceRight()**

The reduceRight() method applies a function against an accumulator and each value of the array (from right-to-left) to reduce it to a single value.

Example

```
const array1 = [[0, 1], [2, 3], [4, 5]].reduceRight(
  (accumulator, currentValue) => accumulator.concat(currentValue)
);
console.log(array1);
```

**22) reverse()**

The reverse() method reverses an array.

Syntax

```
array.reverse()
```

Example

```
var array1 = ['one', 'two', 'three'];
var reversed = array1.reverse();
console.log(reversed);
```

**23) shift()**

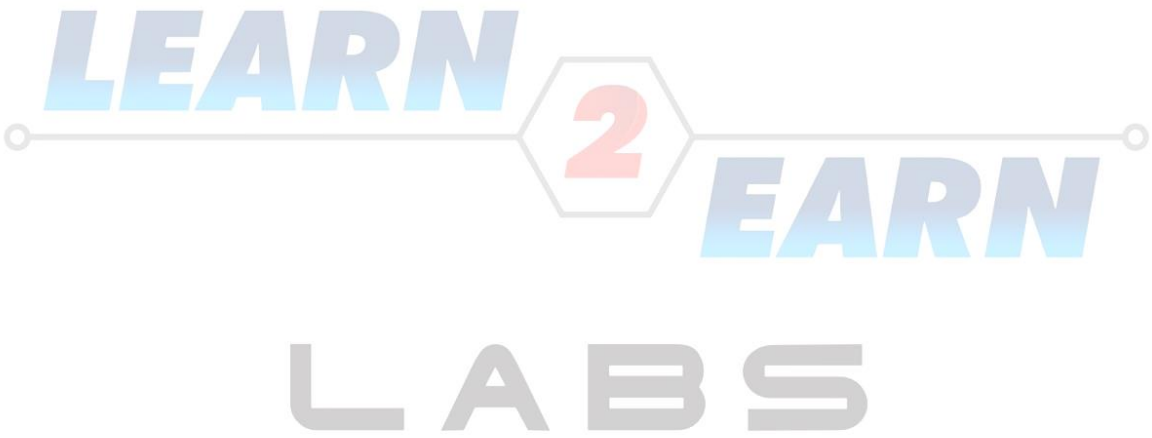
- The shift() method removes the first element from an array and returns that removed element.
- This method changes the length of the array.

Syntax

```
arr.shift()
```

Example

```
var array1 = [1, 2, 3];
var firstElement = array1.shift();
```



**24) slice()**

- The slice() method returns a shallow copy of a portion of an array into a new array object selected from begin to end (end not included).
- The original array will not be modified.
- Not Counts Index.

Syntax

```
arr.slice(begin, end)
```

Example

```
var animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
console.log(animals.slice(2));
// expected output: Array ["camel", "duck", "elephant"]
console.log(animals.slice(2, 4));
// expected output: Array ["camel", "duck"];
```

**25) some()**

- The some() method tests whether at least one element in the array passes the test implemented by the provided function.
- It returns a Boolean value.

Syntax

```
array.some(function(currentValue, index, arr), thisValue)
```

Example

```
function checkAvailability(arr, val) {
    return arr.some(function (arrVal) {
        return val === arrVal;
    });
}
function func() {
    // Original function
    let arr = [2, 5, 8, 1, 4];
     // Checking for condition
    console.log(checkAvailability(arr, 2));
    console.log(checkAvailability(arr, 87));
}
func();
```

**26) sort()**

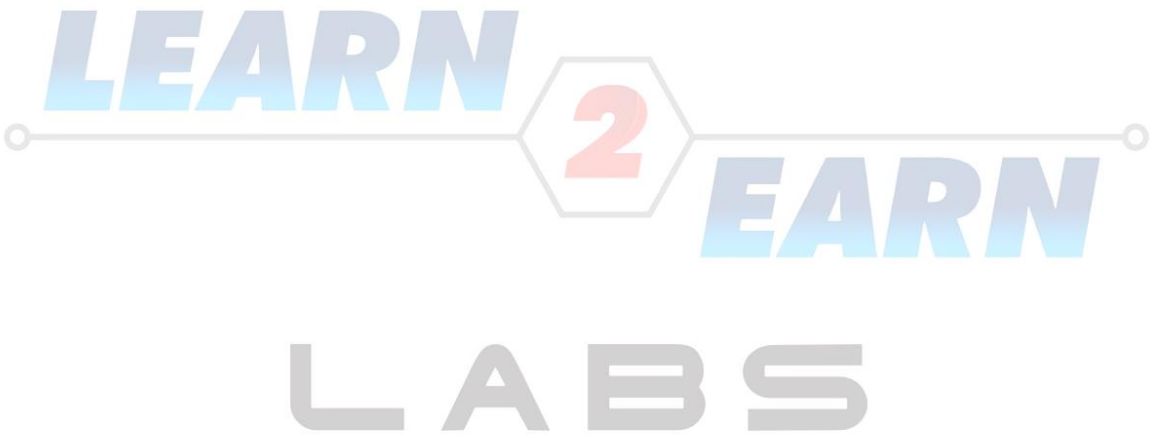
The sort() method sorts the elements of an array.

Syntax

```
arr.sort(arrName)
```

Example

```
var months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);
```



**27) splice()**

The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements.

Syntax

```
arr.splice(target ,no. of elements to delete,"element to insert")
```

Example

```
var months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
console.log(months);
```

**28) toString()**

The toString() method returns a string representing the specified array and its elements.

Syntax

```
arr.toString()
```

Example

```
var array1 = [1, 2, 'a', '1a'];
console.log(array1.toString());
// expected output: "1,2,a,1a"
```

**29) unshift()**

The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.

Syntax

```
array.unshift(item1, item2, ..., itemX)
```

Example

```
var array1 = [1, 2, 3];
console.log(array1.unshift(4, 5));
// expected output: [4,5,1,2,3]
console.log(array1);
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var bac = fruits.unshift("Lemon","Pineapple");
console.log(bac)
```

**30) valueOf()**

The valueOf() method returns the array.

Syntax

```
array.valueOf()
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var v = fruits.valueOf();
console.log(v);
```

**31) values()**

The values() method returns a new Array Iterator object that contains the values for each index in the array.

Syntax

```
arr.values()
```

Example

```
const array1 = ['a', 'b', 'c'];
const iterator = array1.values();
for (const value of iterator) {
  console.log(value); // expected output: "a" "b" "c"
}
```

32) **eval()**

The eval() function evaluates JavaScript code represented as a string.

Syntax

```
eval(string)
```

Example

```
eval("console.log('this is executed by eval()')");
```

Example

```
var result = eval("5+5")
console.log(result)
```

Example

```
var result;
function Sum(val1, val2)
{
   return val1 + val2;
}
eval("result = Sum(5, 5);");
console.log(result);
```

Example

```
var obj = {
  name : "Rohit",
  age :  27
}
var printObj = eval(obj)
console.log(printObj.name)
```