**JavaScript Array Methods**

# JavaScript Array concat()

The concat() method returns a new array by merging two or more values/arrays.

### Example

let primeNumbers = [2, 3, 5, 7]

let evenNumbers = [2, 4, 6, 8]

// join two arrays

let joinedArrays = primeNumbers.concat(evenNumbers);

console.log(joinedArrays);

/\* Output:

[

2, 3, 5, 7,

2, 4, 6, 8

]

\*/

## concat() Syntax

The syntax of the concat() method is:

arr.concat(value1, value2, ..., valueN)

Here, arr is an array.

## concat() Parameters

The concat() method takes in an arbitrary number of arrays and/or values as arguments.

## concat() Return Value

* Returns a newly created array after merging all arrays/values passed in the argument.

The concat() method first creates a new array with the elements of the object on which the method is called. It then sequentially adds arguments or the elements of arguments (for arrays).

# Javascript Array constructor

The constructor property returns the constructor function for the array.

### Example

let languages = ["JavaScript", "Java", "Python"];

let constructor = languages.constructor;

console.log(constructor)

// Output:

// [Function: Array]

## constructor Syntax

The syntax to access the constructor property is:

arr.constructor

Here, arr is an array.

constructor Parameters

The constructor is a property in JavaScript, so it doesn't take any parameters.

## constructor Return Value

* Returns the constructor function for the array.
* For JavaScript arrays, the constructor property returns **function Array() { [native code] }**.

# Javascript Array copyWithin()

The copyWithin() method copies array elements from one position to another in the given array.

### Example

let words = ["apple", "ball", "cat", "dog"];

// copies element from index 0 to index 3

words.copyWithin(3, 0);

// modifies the original array

console.log(words);

// Output:

// [ ''apple'', ''ball'', ''cat'', ''apple'' ]

## copyWithin() Syntax

The syntax of the copyWithin() method is:

arr.copyWithin(target, start, end)

Here, arr is an array.

copyWithin() Parameters

The copyWithin() method can take **three** parameters:

* target - The index position to copy the elements to.
* start (optional) - The index position to start copying elements from. If omitted, it will copy from index **0**.
* end (optional) - The index position to stop copying elements from (end element not included). If omitted, it will copy until the last index.

**Notes:**

* If any of the arguments are negative, the index will be counted backward. For example, **-1** represents the last element, and so on.

## copyWithin() Return Value

* Returns the modified array after copying the elements.

**Notes**: The copyWithin() method:

* overwrites the original array.
* does not change the length of the original array.

# Javascript Array entries()

The entries() method returns a new Array Iterator object containing key/value pairs for each array index.

### Example

// defining an array named alphabets

const alphabets = ["A", "B", "C"];

// array iterator object that contains

// key-value pairs for each index in the array

let iterator = alphabets.entries();

// iterating through key-value pairs in the array

for (let entry of iterator) {

console.log(entry);

}

// Output:

// [ 0, 'A' ]

// [ 1, 'B' ]

// [ 2, 'C' ]

## entries() Syntax

The syntax of the entries() method is:

arr.entries()

Here, arr is an array.

## entries() Parameters

The entries() method does not take any parameters.

## entries() Return Value

* Returns a new Array iterator object.

**Note**: The entries() method does not change the original array.

# Javascript Array every()

The JavaScript Array every() method checks if all the array elements pass the given test function.

The syntax of the every() method is:

arr.every(callback(currentValue), thisArg)

Here, arr is an array.

## every() Parameters

The every() method takes in:

* callback - The function to test for each array element. It takes in:
  + currentValue - The current element being passed from the array.
* thisArg (optional) - Value to use as this when executing callback. By default, it is undefined.

## Return value from every()

* Returns true if all array elements pass the given test function (callback returns a truthy value).
* Otherwise, it returns false.

**Notes**:

* every() does not change the original array.
* every() does not execute callback for array elements without values.

# JavaScript Array fill()

The fill() method returns an array by filling all elements with a specified value.

### Example

// defining an array

var fruits = ['Apple', 'Banana', 'Grape'];

// filling every element of the array with 'Cherry'

fruits.fill("Cherry");

console.log(fruits);

// Output:

// [ 'Cherry', 'Cherry', 'Cherry' ]

## fill() Syntax

The syntax of the fill() method is:

arr.fill(value, start, end)

Here, arr is an array.

## fill() Parameters

The fill() method can take **3** parameters:

* value - Value to fill the array with.
* start (optional) - Start index (default is **0**).
* end (optional) - End index (default is **Array.length**), which is always excluded.

## fill() Return Value

* Returns the modified array, filled with value from start to end.

**Notes:**

* If start or end is negative, indexes are counted backwards.
* Since fill() is a mutator method, it changes the array itself (not a copy) and returns it.

## Example 2: fill() Method with Three Arguments

// array definition

var language = ["JavaScript", "Python", "C", "C++"];

// replacing element of array from index 1 to 3 by 'JavaScript'

language.fill("JavaScript", 1, 3);

// printing the original array

console.log(language);

**Output**

[ 'JavaScript', 'JavaScript', 'JavaScript', 'C++']

Here, we have used the fill() method to fill 'JavaScript' in language from index **1** to **3** (excluding **3**).

So the method just replace the element of **language[1]** and **language[2]** with 'JavaScript'.

# Javascript Array filter()

The filter() method returns a new array with all elements that pass the test defined by the given function.

### Example

let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// function to check even numbers

function checkEven(number) {

if (number % 2 == 0)

return true;

else

return false;

}

// create a new array by filter even numbers from the numbers array

let evenNumbers = numbers.filter(checkEven);

console.log(evenNumbers);

// Output: [ 2, 4, 6, 8, 10 ]

## filter() Syntax

The syntax of the filter() method is:

arr.filter(callback(element), thisArg)

Here, arr is an array.

## filter() Parameters

The filter() method takes in:

* callback - The test function to execute on each array element; returns true if element passes the test, else false. It takes in:
  + element - The current element being passed from the array.
* thisArg (optional) - The value to use as this when executing callback. By default, it is undefined.

## filter() Return Value

* Returns a new array with only the elements that passed the test.

**Notes**:

* filter() does not change the original array.
* filter() does not execute callback for array elements without values.

## Example 1: Filtering out values from Array

const prices = [1800, 2000, null, 3000, 5000, "Thousand", 500, 8000]

function checkPrice(element) {

return element > 2000 && !Number.isNaN(element);

}

let filteredPrices = prices.filter(checkPrice);

console.log(filteredPrices); // [ 3000, 5000, 8000 ]

// using arrow function

let newPrices = prices.filter((price) => (price > 2000 && !Number.isNaN(price)));

console.log(newPrices); // [ 3000, 5000, 8000 ]

**Output**

[ 3000, 5000, 8000 ]

[ 3000, 5000, 8000 ]

Here, all the numbers **less than or equal to 2000**, and all the **non-numeric** values are filtered out.

# JavaScript Array find()

The find() method returns the value of the first array element that satisfies the provided test function.

### Example

let numbers = [1, 3, 4, 9, 8];

// function to check even number

function isEven(element) {

return element % 2 == 0;

}

// get the first even number

let evenNumber = numbers.find(isEven);

console.log(evenNumber);

// Output: 4

## find() Syntax

The syntax of the find() method is:

arr.find(callback(element, index, arr),thisArg)

Here, arr is an array.

## find() Parameters

The find() method takes in:

* callback - Function to execute on each element of the array. It takes in:
  + element - The current element of array.
* thisArg (optional) - Object to use as this inside callback.

## find() Return Value

* Returns the **value** of the **first element** in the array that satisfies the given function.
* Returns undefined if none of the elements satisfy the function.

## Example 1: Using find() method

function isEven(element) {

return element % 2 == 0;

}

let randomArray = [1, 45, 8, 98, 7];

let firstEven = randomArray.find(isEven);

console.log(firstEven); // 8

// using arrow operator

let firstOdd = randomArray.find((element) => element % 2 == 1);

console.log(firstOdd); // 1

**Output**

8

1

# JavaScript Array findIndex()

The findIndex() method returns the index of the first array element that satisfies the provided test function or else returns -1.

### Example

// function that returns odd number

function isOdd(element) {

return element % 2 !== 0;

}

// defining an array of integers

let numbers = [2, 8, 1, 3, 4];

// returns the index of the first odd number in the array

let firstOdd = numbers.findIndex(isOdd);

console.log(firstOdd);

// Output: 2

## findIndex() Syntax

The syntax of the findIndex() method is:

arr.findIndex(callback(element, index, arr),thisArg)

Here, arr is an array.

## findIndex() Parameters

The findIndex() method can take **two** parameters:

* callback - Function to execute on each element of the array. It takes in:
  + element - The current element of array.
* thisArg (optional) - Object to use as this inside callback.

## findIndex() Return Value

* Returns the **index** of the **first element** in the array that satisfies the given function.
* Returns **-1** if none of the elements satisfy the function.

## Example 1: Using findIndex() method

// function that returns even number

function isEven(element) {

return element % 2 == 0;

}

// defining an array of integers

let numbers = [1, 45, 8, 98, 7];

// returns the index of the first even number in the array

let firstEven = numbers.findIndex(isEven);

console.log(firstEven); // 2

**Output**

2

In the above example, we have used the findIndex() method to find the index of the first even number in the numbers array.

# JavaScript Array flat()

The flat() method creates a new array by flattening a nested array up to the specified depth.

### Example

// 3 nested arrays

let numbers = [1, 2, [3, 4, [5, 6, [7, 8]]]];

// reducing nesting by flattening the array to depth 2

let flattenArray = numbers.flat(2);

// new flatten array

console.log(flattenArray);

// Output:

// [ 1, 2, 3, 4, 5, 6, [ 7, 8 ] ]

## flat() Syntax

The syntax of the flat() method is:

arr.flat(depth)

Here, arr is an array.

## flat() Parameters

The flat() method takes a **single** parameter:

* depth - Integer specifying how deep a nested array should be flattened. Its default value is **1**.

## flat() Return Value

* Returns a flatted array with the sub-array elements concatenated into it.

**Notes**: The flat() method:

* does not change the original array.
* removes empty slots in arrays.

## Example 2: flat() Without Default Depth Argument

// nested array

let array1 = [1, [2, 3, 4], 5];

// without passing any depth argument in flat()

let flattenArray = array1.flat();

console.log(flattenArray);

**Output**

[ 1, 2, 3, 4, 5 ]

Here, we have not passed any depth argument in the flat() method.

The default depth argument is **1**, so array1.flat() flattens array1 to depth **1**.

# JavaScript Array flatMap()

The flatMap() method first maps each element of an array using a mapping function, then flattens it into a new array.

### Example

// defining an array

let numbers = [1, 2, 3, 4, 5];

// each element of the array is squared and later flattened

const resultingArray = numbers.flatMap((x) => [x \*\* 2]);

console.log(resultingArray);

// Output:

// [ 1, 4, 9, 16, 25 ]

## flatMap() Syntax

The syntax of the flatMap() method is:

arr.flatMap(callback(currentValue),thisArg)

Here, arr is an array.

## flatMap() Parameters

The flatMap() method can take **two** parameters:

* callback - The function to initially execute on each array element. It takes in:
  + currentValue - The current element being passed from the array.
* thisArg (optional) - Value to use as this when executing callback.

## flatMap() Return Value

* Returns a new array after mapping every element using callback

**Notes**:

* The flatMap() method does not change the original array.
* The flatMap() method is equivalent to array.map().flat().

## Example 1: Using flatMap() Method

// defining an array

let numbers = [1, 2, 3, 4, 5];

// each element of the array is incremented by 1

// and later the array is flattened

let resultingArray = numbers.flatMap((element) => element + 1);

console.log(resultingArray);

**Output**

[ 2, 3, 4, 5, 6 ]

# Javascript Array forEach()

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

### Example

let numbers = [1, 3, 4, 9, 8];

// function to compute square of each number

function computeSquare(element) {

console.log(element \* element);

}

// compute square root of each element

numbers.forEach(computeSquare);

/\* Output:

1

9

16

81

64

\*/

## forEach() Syntax

The syntax of the forEach() method is:

arr.forEach(callback(currentValue), thisArg)

Here, arr is an array.

## forEach() Parameters

The forEach() method takes in:

* callback - The function to execute on every array element. It takes in:
  + currentValue - The current element being passed from the array.
* thisArg (optional) - Value to use as this when executing callback. By default, it is undefined.

## forEach() Return Value

* Returns undefined.

**Notes**:

* forEach() does not change the original array.
* forEach() executes callback once for each array element in order.
* forEach() does not execute callback for array elements without values.

## Example 2: Using thisArg

function Counter() {

this.count = 0;

this.sum = 0;

this.product = 1;

}

Counter.prototype.execute = function (array) {

array.forEach((entry) => {

this.sum += entry;

++this.count;

this.product \*= entry;

}, this)

}

const obj = new Counter();

obj.execute([4, 1, , 45, 8]);

console.log(obj.count); // 4

console.log(obj.sum); // 58

console.log(obj.product); // 1440

**Output**

4

58

1440

Here, we can again see that forEach skips the empty element. thisArg is passed as this inside the definition of the execute method of the Counter object.

# Javascript Array.from()

The from() method creates a new array from any array-like or iterable object.

### Example

// creating a new array from string

let newArray = Array.from("abc");

console.log(newArray);

// Output:

// [ 'a', 'b', 'c' ]

## from() Syntax

The syntax of the from() method is:

Array.from(arraylike, mapFunc, thisArg)

The from() method, being a static method, is called using the Array class name.

## from() Parameters

The from() method can take **three** parameters:

* arraylike - Array-like or iterable object to convert to an array.
* mapFunc (optional) - Map function that is called on each element.
* thisArg (optional) - Value to use as this when executing mapFunc.

**Note**: Array.from(obj, mapFunc, thisArg) is equivalent to Array.from(obj).map(mapFunc, thisArg).

## from() Return Value

* Returns a new Array instance.

**Note**: This method can create an array from:

* array-like objects - The objects that have length property and have indexed elements like String.
* Iterable objects like Map or Set.

## Example 2: from() Method with a Set

// defining a Set

let set = new Set(["JavaScript", "Python", "Go", "Python"]);

// creating an array from the given set

let result = Array.from(set);

console.log(result);

Run Code

**Output**

[ 'JavaScript', 'Python', 'Go' ]

# JavaScript Array includes()

The includes() method checks if an array contains a specified element or not.

### Example

// defining an array

let languages = ["JavaScript", "Java", "C"];

// checking whether the array contains 'Java'

let check = languages.includes("Java");

console.log(check);

// Output: true

## includes() Syntax

The syntax of the includes() method is:

arr.includes(valueToFind, fromIndex)

Here, arr is an array.

## includes() Parameters

The includes() method can take **two** parameters:

* searchValue- The value to search for.
* fromIndex (optional) - The position in the array at which to begin the search. By default, it is **0**.

**Note:** For negative values, the search starts from **array.length + fromIndex** (Counting from backward). For example, **-1** represents the last element.

## includes() Return Value

The includes() method returns:

* true if searchValue is found anywhere within the array
* false if searchValue is not found anywhere within the array

## Example 1: Using includes() method

let languages = ["JavaScript", "Java", "C", "C++"];

// checking whether the array contains 'C'

let check1 = languages.includes("C");

console.log(check1); // true

// checking whether the array contains 'Ruby'

let check2 = languages.includes("Ruby");

console.log(check2); // false

Run Code

**Output**

true

false

In the above example, we have used the includes() method to check whether the languages array contains elements 'C' and 'Ruby'.

languages.includes("C") returns true since the array contains 'C' and languages.includes("Ruby") returns false since the array does not contain 'Ruby'.

## Example 2: includes() for Case-Sensitive Search

The includes() method is case sensitive. For example:

let languages = ["JavaScript", "Java", "C", "Python"];

// checking whether the array contains 'Python'

let check1 = languages.includes("Python");

console.log(check1); // true

// checking whether the array contains 'python'

let check2 = languages.includes("python");

console.log(check2); // false

Run Code

**Output**

true

false

Here the includes() method returns true for searchValue- 'Python' and false for 'python'.

This is because the method is case sensitive and it treats 'Python' and 'python' as two different strings.

# JavaScript Array indexOf()

The indexOf() method returns the first index of occurance of an array element, or **-1** if it is not found.

### Example

let languages = ["Java", "JavaScript", "Python", "JavaScript"];

// get the index of the first occurrence of "JavaScript"

let index = languages.indexOf("JavaScript");

console.log(index);

// Output: 1

The indexOf() method returns the first index of occurance of an array element, or **-1** if it is not found.

### Example

let languages = ["Java", "JavaScript", "Python", "JavaScript"];

// get the index of the first occurrence of "JavaScript"

let index = languages.indexOf("JavaScript");

console.log(index);

// Output: 1

## indexOf() Syntax

The syntax of the indexOf() method is:

arr.indexOf(searchElement, fromIndex)

Here, arr is an array.

## indexOf() Parameters

The indexOf() method takes in:

* searchElement - The element to locate in the array.
* fromIndex (optional) - The index to start the search at. By default, it is **0**.

## indexOf() Return Value

* Returns the first index of the element in the array if it is present at least once.
* Returns **-1** if the element is not found in the array.

**Note:** indexOf() compares searchElement to elements of the Array using **strict equality** (similar to triple-equals operator or ===).

## Example 1: Using indexOf() method

var priceList = [10, 8, 2, 31, 10, 1, 65];

// indexOf() returns the first occurance

var index1 = priceList.indexOf(31);

console.log(index1); // 3

var index2 = priceList.indexOf(10);

console.log(index2); // 0

// second argument specifies the search's start index

var index3 = priceList.indexOf(10, 1);

console.log(index3); // 4

// indexOf returns -1 if not found

var index4 = priceList.indexOf(69.5);

console.log(index4); // -1

**Output**

3

0

4

-1

**Notes:**

* If **fromIndex >= array.length**, array is not searched and **-1** is returned.
* If **fromIndex < 0**, the index is calculated backward. For example, **-1** denotes the last element's index and so on.

# Javascript Array isArray()

The isArray() method checks whether the passed argument is an array or not.

### Example

let numbers = [1, 2, 3, 4];

// checking whether numbers is an array or not

console.log(Array.isArray(numbers));

let text = "JavaScript";

// checking whether text is an array or not

console.log(Array.isArray(text));

// Output:

// true

// false

## isArray() Syntax

The syntax of the isArray() method is:

Array.isArray(value)

The isArray() method, being a static method, is called using the Array class name.

## isArray() Parameters

The isArray() method takes a **single** parameter:

* value - The value to be checked.

## isArray() Return Value

The isArray() method returns:

* true if the passed value is Array
* false if the passed value is not Array

**Note**: This method always returns false for TypedArray instances.

## Example 2: isArray() to Check Other Data Types

// passing an empty array []

console.log(Array.isArray([])); // true

// we have created an array with element 7 and

// passed that value to isArray()

console.log(Array.isArray(new Array(7))); // true

// passing a boolean value

console.log(Array.isArray(true)); // false

// passing undefined

console.log(Array.isArray(undefined)); // false

// not passing any argument in isArray()

console.log(Array.isArray()); // false

Run Code

**Output**

true

true

false

false

false

# Javascript Array join()

The join() method returns a new string by concatenating all of the elements in an array, separated by a specified separator.

### Example

let message = ["JavaScript", "is", "fun."];

// join all elements of array using space

let joinedMessage = message.join(" ");

console.log(joinedMessage);

// Output: JavaScript is fun.

## join() Syntax

The syntax of the join() method is:

arr.join(separator)

Here, arr is an array.

## join() Parameters

The join() method takes in:

* separator (optional) - A string to separate each pair of adjacent elements of the array. By default, it is comma ,.

## join() Return Value

* Returns a [String](https://www.programiz.com/javascript/string) with all the array elements joined by separator.

**Notes**:

* The join() method does not change the original array.
* Elements like undefined, null, or empty array have an empty string representation.

## Example: Using join() method

var info = ["Terence", 28, "Kathmandu"];

var info\_str = info.join(" | ");

// join() does not change the original array

console.log(info); // [ 'Terence', 28, 'Kathmandu' ]

// join() returns the string by joining with separator

console.log(info\_str); // Terence | 28 | Kathmandu

// empty argument = no separator

var collection = [3, ".", 1, 4, 1, 5, 9, 2];

console.log(collection.join("")); // 3.141592

var random = [44, "abc", undefined];

console.log(random.join(" and ")); // 44 and abc and

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Output**

[ 'Terence', 28, 'Kathmandu' ]

Terence | 28 | Kathmandu

3.141592

44 and abc and

Here, we can see that the join() method converts all the array elements into a string and separates each element by the specified separator.

# JavaScript Array keys()

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

### Example

let alphabets = ["A", "B", "C"];

// returns an Array Iterator object that contains the keys

let iterator = alphabets.keys();

// looping through the Iterator object

for (let key of iterator) {

console.log(key);

}

// Output:

// 0

// 1

// 2

## keys() Syntax

The syntax of the keys() method is:

arr.keys()

Here, arr is an array.

## keys() Parameters

The keys() method does not take any parameters.

## keys() Return Value

* Returns a new Array iterator object.

**Notes**: The keys() method **does not**:

* change the original array.
* ignore empty array elements.

## Example 1: Using keys() Method

let languages = ["JavaScript", "Java", "C++", "Python"];

// returns an Array Iterator Object that contains keys

let iterator = languages.keys();

// looping through the iterator object

for (let key of iterator) {

console.log(key);

}

Run Code

**Output**

0

1

2

3

In the above example, we have used the keys() method to find out the key of each element in the languages array.

Here, languages.keys() returns an Array Iterator object whose value is stored in iterator.

And finally, we have looped through iterator that prints the key for each element of language.

# JavaScript Array lastIndexOf()

The lastIndexOf() method returns the index of the last occurrence of a specified element in the array.

### Example

let priceList = [10, 8, 2, 31, 10, 31, 65];

// finding the index of the last occurence of 31

let lastIndex = priceList.lastIndexOf(31);

console.log(lastIndex);

// Output: 5

## lastIndexOf() Syntax

The syntax of the lastIndexOf() method is:

arr.lastIndexOf(searchElement, fromIndex)

Here, arr is an array.

## lastIndexOf() Parameters

The lastIndexOf() method can take **two** parameters:

* searchElement - The element to locate in the array.
* fromIndex (optional) - The index to start searching backwards. By default it is **array.length - 1**.

## lastIndexOf() Return Value

The lastIndexOf() method returns:

* the last index of the element in the array if it is present at least once.
* **-1** if the element is not found in the array.

**Note:** lastIndexOf() compares searchElement to elements of the Array using **strict equality** (similar to triple-equals operator or ===).

## Example 1: Using lastIndexOf() Method

let alphabets = ["a", "b", "c", "a", "d"];

// finding the index of the last occurence of 'a'

let lastIndex1 = alphabets.lastIndexOf("a");

console.log(lastIndex1);

// finding the index of the last occurence of 'e'

let lastIndex2 = alphabets.lastIndexOf("e");

console.log(lastIndex2);

Run Cod

**Output**

3

-1

In the above example, we have used the lastIndexOf() method to find the index of the last occurrence of 'a' and 'e'.

The last occurrence of 'a' is at index **3** in alphabets so alphabets.lastIndexOf("a") returns **3**.

alphabets.lastIndexOf("e") returns **-1** because the array does not contain 'e'.

# JavaScript Array length

The length property returns or sets the number of elements in an array.

### Example

let city = ["California", "Barcelona", "Paris", "Kathmandu"];

// find the length of the city array

let len = city.length;

console.log(len);

// Output: 4

## length Syntax

The syntax to access the length property is:

arr.length

Here, arr is an array.

# JavaScript Array map()

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

### Example

let numbers = [2, 4, 6, 8, 10];

// function to return the square of a number

function square(number) {

return number \* number;

}

// apply square() function to each item of the numbers list

let square\_numbers = numbers.map(square);

console.log(square\_numbers);

// Output: [ 4, 16, 36, 64, 100 ]

## map() Syntax

The syntax of the map() method is:

arr.map(callback(currentValue), thisArg)

Here, arr is an array.

## map() Parameters

The map() method takes in:

* callback - The function called for every array element. Its return values are added to the new array. It takes in:
  + currentValue - The current element being passed from the array.
* thisArg (optional) - Value to use as this when executing callback. By default, it is undefined.

## map() Return Value

* Returns a new array with elements as the return values from the callback function for each element.

**Notes**:

* map() does not change the original array.
* map() executes callback once for each array element in order.
* map() does not execute callback for array elements without values.

## Example 1: Mapping array elements using custom function

const prices = [1800, 2000, 3000, 5000, 500, 8000];

let newPrices = prices.map(Math.sqrt);

// [ 42.42640687119285, 44.721359549995796, 54.772255750516614,

// 70.71067811865476, 22.360679774997898, 89.44271909999159 ]

console.log(newPrices);

// custom arrow function

const string = "JavaScript";

const stringArr = string.split(''); // array with individual string character

let asciiArr = stringArr.map(x => x.charCodeAt(0));

// map() does not change the original array

console.log(stringArr); // ['J', 'a', 'v', 'a','S', 'c', 'r', 'i', 'p', 't']

console.log(asciiArr); // [ 74, 97, 118, 97, 83, 99, 114, 105, 112, 116 ]

**Output**

[

42.42640687119285,

44.721359549995796,

54.772255750516614,

70.71067811865476,

22.360679774997898,

89.44271909999159

]

[

'J', 'a', 'v', 'a',

'S', 'c', 'r', 'i',

'p', 't'

]

[

74, 97, 118, 97,

83, 99, 114, 105,

112, 116

]

# JavaScript Array of() Method

The of() method creates a new Array instance from the given arguments.

### Example

// creating an array named alphabets with elements A,B,C

let alphabets = Array.of("A", "B", "C");

// display contents of alphabet'

console.log(alphabets):

// Output: [ 'A', 'B', 'C' ]

[Run Code](https://www.programiz.com/javascript/online-compiler)

## of() Syntax

The syntax of the of() method is:

Array.of(element0, element1, ..., elementN)

The of() method, being a static method, is called using the Array class name.

## of() Parameters

The of() method can take n number of parameters:

* n specifies the number of elements inside the new array.

## of() Return Value

* Returns a new Array instance.

# JavaScript Array pop()

The pop() method removes the last element from an array and returns that element.

### Example

let cities = ["Madrid", "New York", "Kathmandu", "Paris"];

// remove the last element

let removedCity = cities.pop();

console.log(cities) // ["Madrid", "New York", "Kathmandu"]

console.log(removedCity); // Paris

## pop() Syntax

The syntax of the pop() method is:

arr.pop()

Here, arr is an array.

## pop() Parameters

The pop() method does not have any parameters.

## pop() Return Value

* Removes the last element from array and returns that value.
* Returns undefined if the array is empty.

**Notes**:

* This method changes the original array and its length.
* To remove the first element of an array, use the [JavaScript Array shift()](https://www.programiz.com/javascript/library/array/shift) method.

# JavaScript Array push()

The push() method adds zero or more elements to the end of the array.

### Example

let city = ["New York", "Madrid", "Kathmandu"];

// add "London" to the array

city.push("London");

console.log(city);

// Output: [ 'New York', 'Madrid', 'Kathmandu', 'London' ]

## push() Syntax

The syntax of the push() method is:

arr.push(element1, element2, ..., elementN)

Here, arr is an array.

## push() Parameters

The push() method takes in an arbitrary number of elements to add to the array.

## push() Return Value

* Returns the new (after appending the arguments) length of the array upon which the method was called.

**Notes**:

* This method changes the original array and its length.
* To add elements to the beginning of an array, use the [JavaScript Array unshift()](https://www.programiz.com/javascript/library/array/unshift) method.

# Javascript Array reduce()

The reduce() method executes a reducer function on each element of the array and returns a single output value.

### Example

const message = ["JavaScript ", "is ", "fun."];

// function to join each string elements

function joinStrings(accumulator, currentValue) {

return accumulator + currentValue;

}

// reduce join each element of the string

let joinedString = message.reduce(joinStrings);

console.log(joinedString);

// Output: JavaScript is fun.

[Run Code](https://www.programiz.com/javascript/online-compiler)

## reduce() Syntax

The syntax of the reduce() method is:

arr.reduce(callback(accumulator, currentValue), initialValue)

Here, arr is an array.

reduce() Parameters

The reduce() method takes in:

* callback - The function to execute on each array element (except the first element if no initialValue is provided). It takes in
  + accumulator - It accumulates the callback's return values.
  + currentValue - The current element being passed from the array.
* initialValue (optional) - A value that will be passed to callback() on first call. If not provided, the first element acts as the accumulator on the first call and callback() won't execute on it.

**Note:** Calling reduce() on an empty array without initialValue will throw TypeError.

## reduce() Return Value

* Returns the single value resulting after reducing the array.

**Notes**:

* reduce() executes the given function for each value from left to right.
* reduce() does not change the original array.
* It is almost always safer to provide initialValue.

## Example 1: Sum of All Values of Array

const numbers = [1, 2, 3, 4, 5, 6];

function sum\_reducer(accumulator, currentValue) {

return accumulator + currentValue;

}

let sum = numbers.reduce(sum\_reducer);

console.log(sum); // 21

// using arrow function

let summation = numbers.reduce(

(accumulator, currentValue) => accumulator + currentValue

);

console.log(summation); // 21

**Output**

21

21

## Example 2: Subtracting Numbers in Array

const numbers = [1800, 50, 300, 20, 100];

// subtract all numbers from first number

// since 1st element is called as accumulator rather than currentValue

// 1800 - 50 - 300 - 20 - 100

let difference = numbers.reduce(

(accumulator, currentValue) => accumulator - currentValue

);

console.log(difference); // 1330

const expenses = [1800, 2000, 3000, 5000, 500];

const salary = 15000;

// function that subtracts all array elements from given number

// 15000 - 1800 - 2000 - 3000 - 5000 - 500

let remaining = expenses.reduce(

(accumulator, currentValue) => accumulator - currentValue,

salary

);

console.log(remaining); // 2700

**Output**

1330

2700

This example clearly explains the difference between passing an initialValue and not passing an initialValue.

# Javascript Array reduceRight()

The reduceRight() method reduces the array to a single value by executing a callback function on two values of the array (from right to left).

### Example

let numbers = [1, 2, 3, 4];

// function that adds last two values of the numbers array

function sum\_reducer(accumulator, currentValue) {

return accumulator + currentValue;

}

// returns a single value after reducing the numbers array

let sum = numbers.reduceRight(sum\_reducer);

console.log(sum);

// Output: 10

## reduceRight() Syntax

The syntax of the reduceRight() method is:

arr.reduceRight(callback(accumulator, currentValue), initialValue)

Here, arr is an array.

## reduceRight() Parameters

The reduceRight() method can take **two** parameters:

* callback - The function to execute on each array element. It takes in:
  + accumulator - It accumulates the callback's return values. It is initialValue for the first call if supplied.
  + currentValue - The current element being passed from the array.
* initialValue (optional) - A value that will be passed to callback() on the first call. If not provided, the last element acts as the accumulator on the first call and callback() won't execute on it.

**Note:** Calling reduceRight() on an empty array without initialValue will throw TypeError.

## reduceRight() Return Value

* Returns the value resulting after reducing the array.

**Notes**:

* reduceRight() executes the given function for each value from right to left.
* reduceRight() does not change the original array.
* It is almost always safer to provide initialValue.

## Example 1: Using reduceRight() Method

let numbers = [1, 2, 3, 4, 5, 6];

// function that adds last two values of the numbers array

function sum\_reducer(accumulator, currentValue) {

return accumulator + currentValue;

}

// returns a single value after reducing the numbers array

let sum = numbers.reduceRight(sum\_reducer);

console.log(sum);

Run Code

**Output**

21

# JavaScript Array reverse()

The reverse() method returns the array in reverse order.

### Example

let numbers = [1, 2, 3, 4, 5];

// reversing the numbers array

let reversedArray = numbers.reverse();

console.log(reversedArray);

// Output: [ 5, 4, 3, 2, 1 ]

Run Code

## reverse() Syntax

The syntax of the reverse() method is:

arr.reverse()

Here, arr is an array.

## reverse() Parameters

The reverse() method does not take any parameters.

## reverse() Return Value

* Returns the array after reversing the order of its elements.

**Note:** The reverse() method reverses the order of elements in place, it means the method changes the original array.

## Example 1: Using reverse() Method

let languages = ["JavaScript", "Python", "C++", "Java", "Lua"];

// reversing the order of languages array

let reversedArray = languages.reverse();

console.log("Reversed Array: ", reversedArray);

// modifies the original array

console.log("Original Array: ", languages);

Run Code

**Output**

Reversed Array: [ 'Lua', 'Java', 'C++', 'Python', 'JavaScript' ]

Original Array: [ 'Lua', 'Java', 'C++', 'Python', 'JavaScript' ]

In the above example, we have used the reverse() method to reverse the languages array.

languages.reverse() reverses the order of each element in the array and returns the reversed array.

Since the method modifies the original array, both languages and reversedArray hold the same value.

# JavaScript Array shift()

The shift() method removes the first element from an array and returns that element.

### Example

let languages = ["English", "Java", "Python", "JavaScript"];

// removes the first element of the array

let first = languages.shift()

;

console.log(first);

console.log(languages);

// Output: English

// [ 'Java', 'Python', 'JavaScript' ]

## shift() Syntax

The syntax of the shift() method is:

arr.shift()

Here, arr is an array.

## shift() Parameters

The shift() method does not accept any arguments.

## shift() Return Value

* Removes the first element from array and returns that value.
* Returns undefined if the array is empty.

After removing the element at the **0th** index, it shifts other values to consecutive indexes down.

**Notes**:

* This method changes the original array and its length.
* To remove the last element of an array, use the [JavaScript Array pop()](https://www.programiz.com/javascript/library/array/pop) method.

# JavaScript Array slice()

The slice() method returns a shallow copy of a portion of an array into a new array object.

### Example

let numbers = [2, 3, 5, 7, 11, 13, 17];

// create another array by slicing numbers from index 3 to 5

let newArray = numbers.slice(3, 6);

console.log(newArray);

// Output: [ 7, 11, 13 ]

## slice() Syntax

The syntax of the slice() method is:

arr.slice(start, end)

Here, arr is an array.

## slice() Parameters

The slice() method takes in:

* start (optional) - Starting index of the selection. If not provided, the selection starts at start **0**.
* end (optional) - Ending index of the selection (exclusive). If not provided, the selection ends at the index of the last element.

## slice() Return Value

* Returns a new array containing the extracted elements.

## Example 1: JavaScript slice() method

let languages = ["JavaScript", "Python", "C", "C++", "Java"];

// slicing the array (from start to end)

let new\_arr = languages.slice();

console.log(new\_arr); // [ 'JavaScript', 'Python', 'C', 'C++', 'Java' ]

// slicing from the third element

let new\_arr1 = languages.slice(2);

console.log(new\_arr1); // [ 'C', 'C++', 'Java' ]

// slicing from the second element to fourth element

let new\_arr2 = languages.slice(1, 4);

console.log(new\_arr2); // [ 'Python', 'C', 'C++' ]

Run Code

**Output**

[ 'JavaScript', 'Python', 'C', 'C++', 'Java' ]

[ 'C', 'C++', 'Java' ]

[ 'Python', 'C', 'C++' ]

# Javascript Array some()

The some() method tests whether any of the array elements pass the given test function.

### Example

// a test function: returns an even number

function isEven(element) {

return element % 2 === 0;

}

// defining an array

let numbers = [1, 3, 2, 5, 4];

// checks whether the numbers array contain at least one even number

console.log(numbers.some(isEven));

// Output: true

## some() Syntax

The syntax of the some() method is:

arr.some(callback(currentValue), thisArg)

Here, arr is an array.

## some() Parameters

The some() method can take **two** parameters:

* callback - The function to test for each array element. It takes in:
  + currentValue - The current element being passed from the array.
* thisArg (optional) - Value to use as this when executing callback. By default, it is undefined.

## some() Return Value

* Returns true if an array element passes the given test function (callback returns a truthy value).
* Otherwise, it returns false.

**Notes**: The some() method does not:

* change the original array.
* execute callback for array elements without values.

# JavaScript Array sort()

The sort() method sorts the items of an array in a specific order (ascending or descending).

### Example

let city = ["California", "Barcelona", "Paris", "Kathmandu"];

// sort the city array in ascending order

let sortedArray = city.sort();

console.log(sortedArray);

// Output: [ 'Barcelona', 'California', 'Kathmandu', 'Paris' ]

[Run Code](https://www.programiz.com/javascript/online-compiler)

## sort() Syntax

The syntax of the sort() method is:

arr.sort(compareFunction)

Here, arr is an array.

## sort() Parameters

The sort() method takes in:

* compareFunction (optional) - It is used to define a custom sort order.

## sort() Return Value

* Returns the array after sorting the elements of the array in place (meaning that it changes the original array and no copy is made).

## Example 2: Sorting using Custom Function

When compareFunction is passed,

* All non-undefined array elements are sorted according to the return value of compareFunction.
* All undefined elements are sorted to the end of the array and compareFunction is not called for them.

Suppose we want to sort the above names array such that the longest name comes last, rather than sorting it alphabetically. We can do it in the following way:

// custom sorting an array of strings

var names = ["Adam", "Jeffrey", "Fabiano", "Danil", "Ben"];

function len\_compare(a, b){

return a.length - b.length;

}

// sort according to string length

names.sort(len\_compare);

console.log(names);

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Output**

[ 'Ben', 'Adam', 'Danil', 'Jeffrey', 'Fabiano' ]

Here, the sorting is based on the logic a.length - b.length. It basically means that the item with shorter length will appear at the beginning of the Array.

# JavaScript Array splice()

The splice() method returns an array by changing (adding/removing) its elements in place.

### Example

let prime\_numbers = [2, 3, 5, 7, 9, 11];

// replace 1 element from index 4 by 13

let removedElement = prime\_numbers.splice(4, 1, 13);

console.log(removedElement);

console.log(prime\_numbers);

// Output: [ 9 ]

// [ 2, 3, 5, 7, 13, 11 ]

## spice() Syntax

The syntax of the splice() method is:

arr.splice(start, deleteCount, item1, ..., itemN)

Here, arr is an array.

## splice() Parameters

The splice() method takes in:

* start - The index from where the array is changed.
* deleteCount (optional) - The number of items to remove from start.
* item1, ..., itemN (optional) - The elements to add to the start index. If not specified, splice() will only remove elements from the array.

## splice() Return Value

* Returns an array containing the deleted elements.

**Note:** The splice() method changes the original array.

## Example 1: Using splice() method

let languages = ["JavaScript", "Python", "Java", "Lua"];

// replacing "Java" & "Lua" with "C" & "C++"

let removed = languages.splice(2, 2, "C", "C++");

console.log(removed); // [ 'Java', 'Lua' ]

console.log(languages); // [ 'JavaScript', 'Python', 'C', 'C++' ]

// adding elements without deleting existing elements

let removed1 = languages.splice(1, 0, "Java", "Lua");

console.log(removed1); // []

console.log(languages); // [ 'JavaScript', 'Java', 'Lua', 'Python', 'C', 'C++' ]

// removing 3 elements

let removed2 = languages.splice(2, 3);

console.log(removed2); // [ 'Lua', 'Python', 'C' ]

console.log(languages); // [ 'JavaScript', 'Java', 'C++' ]

**Output**

[ 'Java', 'Lua' ]

[ 'JavaScript', 'Python', 'C', 'C++' ]

[]

[ 'JavaScript', 'Java', 'Lua', 'Python', 'C', 'C++' ]

[ 'Lua', 'Python', 'C' ]

[ 'JavaScript', 'Java', 'C++' ]

# Javascript Array toLocaleString()

The toLocaleString() method returns a string representing the elements of the array in a particular locale.

### Example

let array1 = ["Nepal", 1];

// returns string representation of array

let stringFromArray = array1.toLocaleString();

console.log(stringFromArray);

// Output:

// Nepal,1

## toLocaleString() Syntax

The syntax of the toLocaleString() method is:

arr.toLocaleString(locales, options)

Here, arr is an array.

## toLocaleString() Parameters

The toLocaleString() method can take **two** parameters:

* locales (optional) - A convention or formatting based on particular geography.
* options (optional) - An object with configuration properties.

## toLocaleString() Return Value

* Returns a string representing the elements of the array.

**Note**: This method converts each array element to Strings using their toLocaleString methods and separates them by a comma.

## Example 1: Using toLocaleString() Method

let o = [1, "JavaScript", new Date()];

// returns string representation of array

let stringFromArray = array1.toLocaleString();

console.log(stringFromArray);

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Output**

1,JavaScript,5/9/2022, 2:11:22 PM

In the above example, we have used the toLocaleString() method to convert array1 to a string representing its elements.

We have created array1 with three elements: 1, 'JavaScript' and new Date() where the third element creates a Date object.

array1.toLocaleString() returns the string representation of these elements i.e. 1,JavaScript,5/9/2022, 2:11:22 PM which is separated by locale-specific string like comma.

# Javascript Array toString()

The toString() method returns a string formed by the elements of the given array.

### Example

// defining an array

let items = ["JavaScript", 1, "a", 3];

// returns a string with elements of the array separated by commas

let itemsString = items.toString();

console.log(itemsString);

// Output:

// JavaScript,1,a,3

## toString() Syntax

The syntax of the toString() method is:

arr.toString()

Here, arr is an array.

## toString() Parameters

The toString() method does not take any parameters.

## toString() Return Value

* Returns a string representing the values of the array separated by a comma

**Notes**:

* The toString() method does not change the original array.
* Elements like undefined, null, or empty array, have an empty string representation.

## Example 1: Using toString() Method

let info = ["Terence", 28, "Kathmandu"];

// returns the string representation of the info array

let info\_str = info.toString();

console.log(info\_str);

// toString() does not change the original array

console.log(info);

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Output**

Terence,28,Kathmandu

[ 'Terence', 28, 'Kathmandu' ]

In the above example, we have used the toString() method to convert all the elements of the info array into a string.

info.toString() returns the string representation of info which is Terence,28,Kathmandu.

Since the method does not change the original array, the info array holds the same original value.

# JavaScript Array unshift()

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

### Example

let languages = ["Java", "Python", "C"];

// add "JavaScript" at the beginning of the array

languages.unshift("JavaScript");

console.log(languages);

// Output: [ 'JavaScript', 'Java', 'Python', 'C' ]

## unshift() Syntax

The syntax of the unshift() method is:

arr.unshift(element1, element2, ..., elementN)

Here, arr is an array.

## unshift() Parameters

The unshift() method takes in an arbitrary number of elements to add to the array.

## unshift() Return Value

* Returns the new (after adding arguments to the beginning of array) length of the array upon which the method was called.

**Notes**:

* This method changes the original array and its length.
* To add elements to the end of an array, use the [JavaScript Array push()](https://www.programiz.com/javascript/library/array/push) method.

# JavaScript Array values()

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

### Example

let languages = ["JavaScript", "Java", "C++"];

// returns an Array Iterator object that contain values

let iteratorObject = languages.values();

// looping through iterator

for (let value of iteratorObject) {

console.log(value);

}

// Output:

// JavaScript

// Java

// C++

[Run Code](https://www.programiz.com/javascript/online-compiler)

## values() Syntax

The syntax of the values() method is:

arr.values()

Here, arr is an array.

## values() Parameters

The values() method does not take any parameters.

## values() Return Value

* Returns a new Array iterator object.

**Notes**: The value() method does not change the original array.

# JavaScript ES6

JavaScript **ES6** (also known as **ECMA(** **European Computer Manufacturers Association)Script 2015** or **ECMAScript 6**) is the newer version of JavaScript that was introduced in 2015.

[ECMAScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources) is the standard that JavaScript programming language uses. ECMAScript provides the specification on how JavaScript programming language should work.

## JavaScript let

JavaScript let is used to declare variables. Previously, variables were declared using the var keyword.

The variables declared using let are **block-scoped**. This means they are only accessible within a particular block. For example,

// variable declared using let

let name = 'Sara';

{

// can be accessed only inside

let name = 'Peter';

console.log(name); // Peter

}

console.log(name); // Sara

## JavaScript let Vs var

Here's the overview of the differences between let and var.

|  |  |
| --- | --- |
| let | var |
| let is block-scoped. | var is function scoped. |
| let does not allow to redeclare variables. | var allows to redeclare variables. |
| Hoisting does not occur in let. | Hoisting occurs in var. |

## JavaScript const

The const statement is used to declare constants in JavaScript. For example,

// name declared with const cannot be changed

const name = 'Sara';

Once declared, you cannot change the value of a const variable.