Array.prototype.reduce()

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

Syntax

```
arr.reduce(callback[, initialValue])
```

Parameters

`callback`

Function to execute on each value in the array, taking four arguments:

`previousValue`

The value previously returned in the last invocation of the callback, or `initialValue`, if supplied. (See below.)

`currentValue`

The current element being processed in the array.

`currentIndex`

The index of the current element being processed in the array.

`array`

The array `reduce` was called upon.

`initialValue`

Optional. Value to use as the first argument to the first call of the `callback`.

Description

`reduce` executes the `callback` function once for each element present in the array, excluding holes in the array, receiving four arguments:

- `previousValue`
- `currentValue`
- `currentIndex`
- `array`

The first time the callback is called, `previousValue` and `currentValue` can be one of two values. If `initialValue` is provided in the call to `reduce`, then `previousValue` will be equal to `initialValue` and `currentValue` will be equal to the first value in the array. If no `initialValue` was provided, then `previousValue` will be equal to the first value in the array and `currentValue` will be equal to the second.

If the array is empty and no `initialValue` was provided, `TypeError` would be thrown. If the array has only one element (regardless of position) and no `initialValue` was provided, or if `initialValue` is provided but the array is empty, the solo value would be returned without calling `callback`.

Suppose the following use of `reduce` occurred:

```
[0, 1, 2, 3, 4].reduce(function(previousValue, currentValue, currentIndex, array) {
  return previousValue + currentValue;
});
```

The callback would be invoked four times, with the arguments and return values in each call being as follows:

	previousValue	currentValue	currentIndex	array	return value

first call	0	1	1	[0, 1, 2, 3, 4]	1
second call	1	2	2	[0, 1, 2, 3, 4]	3
third call	3	3	3	[0, 1, 2, 3, 4]	6
fourth call	6	4	4	[0, 1, 2, 3, 4]	10

The value returned by `reduce` would be that of the last callback invocation (`10`).

You can also provide an `Arrow Function` in lieu of a full function. The code below will produce the same output as the code above:

```
[0, 1, 2, 3, 4].reduce( (prev, curr) => prev + curr );
```

If you were to provide an initial value as the second argument to `reduce`, the result would look like this:

```
[0, 1, 2, 3, 4].reduce(function(previousValue, currentValue, currentIndex, array) {
  return previousValue + currentValue;
}, 10);
```

	previousValue	currentValue	currentIndex	array	return value
first call	10	0	0	[0, 1, 2, 3, 4]	10
second call	10	1	1	[0, 1, 2, 3, 4]	11
third call	11	2	2	[0, 1, 2, 3, 4]	13
fourth call	13	3	3	[0, 1, 2, 3, 4]	16
fifth call	16	4	4	[0, 1, 2, 3, 4]	20

The final call return value (`20`) is *returned* as a result of the reduce function

Examples

Sum all the values of an array

```
var total = [0, 1, 2, 3].reduce(function(a, b) {
  return a + b;
});
// total == 6
```

Flatten an array of arrays

```
var flattened = [[0, 1], [2, 3], [4, 5]].reduce(function(a, b) {
  return a.concat(b);
}, []);
// flattened is [0, 1, 2, 3, 4, 5]
```

Polyfill

`Array.prototype.reduce` was added to the ECMA-262 standard in the 5th edition; as such it may not be present in all implementations of the standard. You can work around this by inserting the following code at the beginning of your scripts, allowing use of `reduce` in implementations which do not natively support it.

```
// Production steps of ECMA-262, Edition 5, 15.4.4.21
// Reference: http://es5.github.io/#x15.4.4.21
if (!Array.prototype.reduce) {
  Array.prototype.reduce = function(callback /*, initialValue*/) {
    'use strict';
    if (this == null) {
      throw new TypeError('Array.prototype.reduce called on null or undefined');
    }
```

```
    if (typeof callback !== 'function') {
      throw new TypeError(callback + ' is not a function');
    }
    var t = Object(this), len = t.length >>> 0, k = 0, value;
    if (arguments.length == 2) {
      value = arguments[1];
    } else {
      while (k < len && !(k in t)) {
        k++;
      }
      if (k >= len) {
        throw new TypeError('Reduce of empty array with no initial value');
      }
      value = t[k++];
    }
    for (; k < len; k++) {
      if (k in t) {
        value = callback(value, t[k], k, t);
      }
    }
    return value;
  };
}
```

Specifications

Specification	Status	Comment
ECMAScript 5.1 (ECMA-262) The definition of 'Array.prototype.reduce' in that specification.	ST Standard	Initial definition. Implemented in JavaScript 1.8.
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Array.prototype.reduce' in that specification.	ST Standard	

Browser compatibility

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	(Yes)	3.0 (1.9)	9	10.5	4.0

See also

- `Array.prototype.reduceRight()`