# Array.prototype.forEach() - JavaScript | MDN

## Parámetros

`callback`

`currentValue`

El elemento actual siendo procesado en el array.

`index` Opcional

El índice del elemento actual siendo procesado en el array.

`array` Opcional

El vector en el que `forEach()` esta siendo aplicado.

`thisArg` Opcional

Valor que se usará como `this` cuando se ejecute el `callback`.

## Descripción

`forEach()` ejecuta la función `callback` una vez por cada elemento presente en el array en orden ascendente. No es invocada para índices que han sido eliminados o que no hayan sido inicializados (Ej. sobre arrays `sparse`)

`callback` es invocada con tres argumentos:

1. el valor del elemento
2. el índice del elemento
3. el array que está siendo recorrido

Si un parámetro `thisArg` es proporcionado a `forEach`, será usado como el valor `this` para cada invocación de `callback` como si se llamara a `callback.call(thisArg, element, index, array)`. Si `thisArg` es `undefined` o `null`, el valor `this` dentro de la función depende si la función está o no en modo estricto (valor pasado si está en modo estricto, objeto global si está en modo no-estricto).

El rango de elementos procesados por `forEach()` se establece antes de la primera invocación del `callback`. Los elementos que sean añadidos al vector después de que inicie la llamada a `forEach` no serán visitados por `callback`. Si los valores de los elementos existentes en el vector son modificados, el valor pasado al `callback` será el valor al momento de que forEach los visite; no se evaluarán los elementos borrados antes de ser visitados por `forEach`.

`forEach()` ejecuta la función `callback` una vez por cada elemento del array; a diferencia de `map()` o `reduce()` este siempre devuelve el valor `undefined` y no es encadenable. El típico uso es ejecutar los efectos secundarios al final de la cadena.

`foreach()` no muta/modifica el array desde el que es llamado (aunque `callback`, si se invoca, podría hacerlo).

**Nota: Nota :** No hay forma de detener o cortar un bucle `forEach` que no sea lanzar una excepción. Si necesita dicho comportamiento, el método `.forEach()` es la herramienta equivocada, use una simple iteración en su lugar. Si está probando los elementos del array para un predicado y necesita devolver un valor boleano, puede usar `every()` o `some()` en su lugar.

## Imprimiendo el contenido de un `array`

El siguiente código imprime una línea por cada elemento en un array:

```
function logArrayElements(element, index, array) {
    console.log("a[" + index + "] = " + element);
}
// Nótese que se evita el 2° índice ya que no hay
ningún elemento en esa posición del array
[2, 5, , 9].forEach(logArrayElements);
// salida:
// a[0] = 2
// a[1] = 5
// a[2] = 9
```

Copy to Clipboard

## Usando `thisArg`

El siguiente ejemplo actualiza las propiedades del objeto por cada entrada en el array:

```
function Counter() {
  this.sum = 0;
  this.count = 0;
}
Counter.prototype.add = function(array) {
  array.forEach(function(entry) {
    this.sum += entry;
    ++this.count;
  }, this);
  // ^---- Note
};
```

```
var obj = new Counter();
obj.add([2, 5, 9]);
obj.count
// 3
obj.sum
// 16
```

Copy to Clipboard

Nota: Dado que el parámetro `thisArg` (this) se referencia en el `forEach()`, será pasado al `callback` cuando se invoque, para utilizarse como su valor `this`.

## Ejemplo: Función que copia objetos

El siguiente código crea una copia de un objeto dado. Hay diferentes formas de crear una copia de un objeto, ésta es sólo una de ellas y sirve para explicar cómo funciona `Array.prototype.forEach` utilizando funciones `Object.*` de ECMAScript 5.

```
function copy(o){
  var copy = Object.create( Object.getPrototypeOf(o) );
  var propNames = Object.getOwnPropertyNames(o);

  propNames.forEach(function(name){
    var desc = Object.getOwnPropertyDescriptor(o,
name);
    Object.defineProperty(copy, name, desc);
  });

  return copy;
}
```

```
var o1 = {a:1, b:2};
var o2 = copy(o1); // o2 ahora se parece a o1
```

Copy to Clipboard

## Si el array se modifica durante la iteración, otros elementos pueden ser omitidos.

El siguiente ejemplo muestra por consola "uno", "dos", "cuatro". Cuando se alcanza el registro que contiene el valor "dos", el primer registro del array se desplaza, lo que hace que los registros restantes se muevan una posición. Debido a que el elemento "cuatro" está ahora en una posición anterior en el array, "tres" se omitirá. `forEach()` no hace una copia del array antes de iterar.

```
var words = ['uno', 'dos', 'tres', 'cuatro'];
words.forEach(function(word) {
  console.log(word);
  if (word === 'dos') {
    words.shift();
  }
});
// uno
// dos
// cuatro
```

Copy to Clipboard

## Polyfill

`forEach` se agregó de manera reciente al estándar ECMA-262; así que puede no estar presente en otras implementaciones del estándar. Se puede asegurar el uso del forEach con tan solo agregar el siguiente código al inicio de los scripts, permitiendo así el uso de forEach en implementaciones que no lo soportan de manera nativa. El algoritmo es el mismo que se especifica en la quinta versión de ECMA-262, asumiendo que `Object` y `TypeError` tienen sus valores originales y que callback.call evalúa el valor original de `Function.prototype.call()`.

```
// Production steps of ECMA-262, Edition 5, 15.4.4.18
// Reference: http://es5.github.com/#x15.4.4.18
if (!Array.prototype.forEach) {

  Array.prototype.forEach = function forEach(callback,
thisArg) {
    'use strict';
    var T, k;

    if (this == null) {
      throw new TypeError("this is null or not
defined");
    }

    var kValue,
        // 1. Let O be the result of calling ToObject
passing the |this| value as the argument.
        O = Object(this),

        // 2. Let lenValue be the result of calling the
Get internal method of O with the argument "length".
```

```
        // 3. Let len be ToUint32(lenValue).
        len = O.length >>> 0; // Hack to convert
O.length to a UInt32

    // 4. If IsCallable(callback) is false, throw a
TypeError exception.
    // See: http://es5.github.com/#x9.11
    if ({}.toString.call(callback) !== "[object
Function]") {
      throw new TypeError(callback + " is not a
function");
    }

    // 5. If thisArg was supplied, let T be thisArg;
else let T be undefined.
    if (arguments.length >= 2) {
      T = thisArg;
    }

    // 6. Let k be 0
    k = 0;

    // 7. Repeat, while k < len
    while (k < len) {

      // a. Let Pk be ToString(k).
      //   This is implicit for LHS operands of the in
operator
      // b. Let kPresent be the result of calling the
HasProperty internal method of O with argument Pk.
      //   This step can be combined with c
```

```
      // c. If kPresent is true, then
      if (k in O) {

        // i. Let kValue be the result of calling the
Get internal method of O with argument Pk.
        kValue = O[k];

        // ii. Call the Call internal method of
callback with T as the this value and
        // argument list containing kValue, k, and O.
        callback.call(T, kValue, k, O);
      }
      // d. Increase k by 1.
      k++;
    }
    // 8. return undefined
  };
}
```

Copy to Clipboard