Array.prototype.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.

JavaScript Demo: Array.slice()

```
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"]

console.log(animals.slice(1, 5));
// expected output: Array ["bison", "camel", "duck", "elephant"]

```

Run ›

Reset

Syntax

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

Parameters

`begin` Optional

Zero-based index at which to begin extraction.

A negative index can be used, indicating an offset from the end of the sequence. `slice(-2)` extracts the last two elements in the sequence.

If `begin` is undefined, `slice` begins from index `0`.

If `begin` is greater than the length of the sequence, an empty array is returned.

`end` Optional

Zero-based index *before* which to end extraction. `slice` extracts up to but not including `end`.

For example, `slice(1,4)` extracts the second element through the fourth element (elements indexed 1, 2, and 3).

A negative index can be used, indicating an offset from the end of the sequence. `slice(2,-1)` extracts the third element through the second-to-last element in the

sequence.

If `end` is omitted, `slice` extracts through the end of the sequence (`arr.length`).

If `end` is greater than the length of the sequence, `slice` extracts through to the end of the sequence (`arr.length`).

Return value

A new array containing the extracted elements.

Description

`slice` does not alter the original array. It returns a shallow copy of elements from the original array. Elements of the original array are copied into the returned array as follows:

- For object references (and not the actual object), `slice` copies object references into the new array. Both the original and new array refer to the same object. If a referenced object changes, the changes are visible to both the new and original arrays.
- For strings, numbers and booleans (not `String`, `Number` and `Boolean` objects), `slice` copies the values into the new array. Changes to the string, number or boolean in one array do not affect the other array.

If a new element is added to either array, the other array is not affected.

Examples

Return a portion of an existing array

```
var fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];
var citrus = fruits.slice(1, 3);

// fruits contains ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango']
// citrus contains ['Orange','Lemon']
```

Using `slice`

In the following example, `slice` creates a new array, `newCar`, from `myCar`. Both include a reference to the object `myHonda`. When the color of `myHonda` is changed to purple, both arrays reflect the change.

```
// Using slice, create newCar from myCar.
var myHonda = { color: 'red', wheels: 4, engine: { cylinders: 4, size: 2.2 } };
var myCar = [myHonda, 2, 'cherry condition', 'purchased 1997'];
var newCar = myCar.slice(0, 2);

// Display the values of myCar, newCar, and the color of myHonda
//  referenced from both arrays.
console.log('myCar = ' + JSON.stringify(myCar));
console.log('newCar = ' + JSON.stringify(newCar));
console.log('myCar[0].color = ' + myCar[0].color);
console.log('newCar[0].color = ' + newCar[0].color);

// Change the color of myHonda.
myHonda.color = 'purple';
console.log('The new color of my Honda is ' + myHonda.color);

// Display the color of myHonda referenced from both arrays.
console.log('myCar[0].color = ' + myCar[0].color);
console.log('newCar[0].color = ' + newCar[0].color);
```

This script writes:

```
myCar = [{color: 'red', wheels: 4, engine: {cylinders: 4, size: 2.2}}, 2,
         'cherry condition', 'purchased 1997']
newCar = [{color: 'red', wheels: 4, engine: {cylinders: 4, size: 2.2}}, 2]
myCar[0].color = red
newCar[0].color = red
The new color of my Honda is purple
myCar[0].color = purple
newCar[0].color = purple
```

Array-like objects

`slice` method can also be called to convert Array-like objects / collections to a new Array. You just bind the method to the object. The `arguments` inside a function is an example of an 'array-like object'.

```
function list() {
  return Array.prototype.slice.call(arguments);
}

var list1 = list(1, 2, 3); // [1, 2, 3]
```

Binding can be done with the `.call` function of `Function.prototype` and it can also be reduced using `[].slice.call(arguments)` instead of `Array.prototype.slice.call`. Anyway, it can be simplified using `bind`.

```
var unboundSlice = Array.prototype.slice;
var slice = Function.prototype.call.bind(unboundSlice);

function list() {
  return slice(arguments);
}

var list1 = list(1, 2, 3); // [1, 2, 3]
```

Streamlining cross-browser behavior

Although host objects (such as DOM objects) are not required by spec to follow the Mozilla behavior when converted by `Array.prototype.slice` and IE < 9 does not do so, versions of IE starting with version 9 do allow this. "Shimming" it can allow reliable cross-browser behavior. As long as other modern browsers continue to support this ability, as currently do IE, Mozilla, Chrome, Safari, and Opera, developers reading (DOM-supporting) slice code relying on this shim will not be misled by the semantics; they can safely rely on the semantics to provide the now apparently *de facto* standard behavior. (The shim also fixes IE to work with the second argument of `slice()` being an explicit `null`/`undefined` value as earlier versions of IE also did not allow but all modern browsers, including IE >= 9, now do.)

```
/**
 * Shim for "fixing" IE's lack of support (IE < 9) for applying slice
 * on host objects like NamedNodeMap, NodeList, and HTMLCollection
 * (technically, since host objects have been implementation-dependent,
 * at least before ES2015, IE hasn't needed to work this way).
 * Also works on strings, fixes IE < 9 to allow an explicit undefined
 * for the 2nd argument (as in Firefox), and prevents errors when
 * called on other DOM objects.
 */
(function () {
  'use strict';
  var _slice = Array.prototype.slice;

  try {
    // Can't be used with DOM elements in IE < 9
    _slice.call(document.documentElement);
  } catch (e) { // Fails in IE < 9
```

```
    // This will work for genuine arrays, array-like objects,
    // NamedNodeMap (attributes, entities, notations),
    // NodeList (e.g., getElementsByTagName), HTMLCollection (e.g., childNodes),
    // and will not fail on other DOM objects (as do DOM elements in IE < 9)
    Array.prototype.slice = function(begin, end) {
      // IE < 9 gets unhappy with an undefined end argument
      end = (typeof end !== 'undefined') ? end : this.length;

      // For native Array objects, we use the native slice function
      if (Object.prototype.toString.call(this) === '[object Array]'){
        return _slice.call(this, begin, end);
      }

      // For array like object we handle it ourselves.
      var i, cloned = [],
        size, len = this.length;

      // Handle negative value for "begin"
      var start = begin || 0;
      start = (start >= 0) ? start : Math.max(0, len + start);

      // Handle negative value for "end"
      var upTo = (typeof end == 'number') ? Math.min(end, len) : len;
      if (end < 0) {
        upTo = len + end;
      }

      // Actual expected size of the slice
      size = upTo - start;

      if (size > 0) {
        cloned = new Array(size);
        if (this.charAt) {
          for (i = 0; i < size; i++) {
            cloned[i] = this.charAt(start + i);
          }
        } else {
          for (i = 0; i < size; i++) {
            cloned[i] = this[start + i];
          }
        }
      }

      return cloned;
    };
  }
}());
```

Specifications

Specification	Status	Comment
ECMAScript 3rd Edition (ECMA-262)	ST Standard	Initial definition. Implemented in JavaScript 1.2.
ECMAScript 5.1 (ECMA-262) The definition of 'Array.prototype.slice' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Array.prototype.slice' in that specification.	ST Standard	
ECMAScript Latest Draft (ECMA-262) The definition of 'Array.prototype.slice' in that specification.	D Draft	

Browser compatibility

New compatibility tables are in beta

Basic support													
1	Yes	1	Yes	Yes	Yes	Yes	Yes	Yes	4	Yes	Yes	Yes	Yes

Full support

See also

- `Array.prototype.splice()`
- `Function.prototype.call()`
- `Function.prototype.bind()`