skillcrush

JavaScript Arrays Cheatsheet

Contents:

Add, Remove, & Replace Elements

.push()

Add one or more elements to the *end* of an array.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
typesOfOranges.push("Jaffa");
console.log(typesOfOranges);
// (5) ["Navel", "Clementine", "Valencia", "Bergamot", "Jaffa"]
```

.pop()

Remove a single element from the *end* of an array.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
typesOfOranges.pop();
console.log(typesOfOranges);
// (3) ["Navel", "Clementine", "Valencia"]
```

.unshift()

Add one or more elements to the *beginning* of an array.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
typesOfOranges.unshift("Satsuma", "Sour");
console.log(typesOfOranges);
// (6) ["Satsuma", "Sour", "Navel", "Clementine", "Valencia", "Bergamot"]
```

.shift()

Remove a single element at the *beginning* of an array.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
typesOfOranges.shift();
console.log(typesOfOranges);
// (3) ["Clementine", "Valencia", "Bergamot"]
```

.splice()

Use splice to remove, add, or replace elements in an array from any index position.

To *remove* elements, inside `splice()`, add the index of the first element to remove, then add the quantity of elements to remove:

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
typesOfOranges.splice(1, 2);
console.log(typesOfOranges);
// (2) ["Navel", Bergamot"]
```

To *add* elements (without deleting or replacing), add the index position in `splice()` to add the elements. Leave the second number at "0" because you don't want to delete elements. Next, include the elements you want to add.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
typesOfOranges.splice(3, 0, "Blood", "Seville");
console.log(typesOfOranges);
// (6) ["Navel", "Clementine", "Valencia", "Blood", "Seville", "Bergamot"]
```

To *replace* elements, enter the index position and number of elements to replace, then include the elements you want added instead:

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
typesOfOranges.splice(2, 1, "Hamlin");
console.log(typesOfOranges);
// (6) ["Navel", "Clementine", "Hamlin", "Bergamot"]
```

.slice()

Make a copy of a portion of an array. The first number in `slice()` is the starting index position to copy from. Add the second number in `slice()` to target a specific index range of elements.
Note: The range number doesn't target the element from the starting index position.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];

console.log(typesOfOranges.slice(1));
// (3) ["Clementine", "Valencia", "Bergamot"]
console.log(typesOfOranges.slice(1, 3));
// (2) ["Clementine", "Valencia"]
```

Use the copied elements from `slice()` along with other methods, like `shift()` or `push()`, to create a new array. The original array will remain unchanged.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];

var newCitrus = typesOfOranges.slice(0, 1);
newCitrus.push("Lemon", "Pomelo");

console.log(newCitrus);
// (3) ["Navel", "Lemon", "Pomelo"]
console.log(typesOfOranges);
// (4) ["Navel", "Clementine", "Valencia", "Bergamot"]
```

Identify Length, Elements, & Indices

.length

Identify the number of elements in an array with this property.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];

console.log(typesOfOranges.length);
// 4
```

.includes()

Find out if an array has a particular element. In the console, you'll see a boolean "true" if the array contains the elements, and "false" if it doesn't.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];

console.log(typesOfOranges.includes("Clementine"));
// true
console.log(typesOfOranges.includes("Hamlin"));
// false
```

.indexOf()

Find the index of the *first* occurrence of an element in an array.

```
var numOfOranges = [12, 30, 4, 12, 4, 8, 25, 8];
var num = numOfOranges.indexOf(4);

console.log(num);
// 2
```

Combine & Create New Arrays

.concat()

Concatenate (join) two or more arrays.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia"];
var typesOfGrapefruit = ["Marsh", "Flame", "Ruby Red"];

var citrus = typesOfOranges.concat(typesOfGrapefruit);
console.log(citrus);
// (6) ["Navel", "Clementine", "Valencia", "Marsh", "Flame", "Ruby Red"]
```

spread operator (...)

Instead of using a method to combine arrays, you can add the spread operator (...) before the variable names of the arrays inside another array.

```
var typesOfOranges = ["Navel", "Clementine", "Valencia"];
var typesOfGrapefruit = ["Marsh", "Flame", "Ruby Red"];

var citrus = [...typesOfOranges, ...typesOfGrapefruit];
console.log(citrus);
// (6) ["Navel", "Clementine", "Valencia", "Marsh", "Flame", "Ruby Red"]
```

The spread syntax is also used to pass elements of an array to functions that require separate arguments and cannot accept arrays themselves.

```
var dailySteps = [4350, 8098, 2444, 2536, 9875, 4456, 2453];
console.log(Math.min(...dailySteps));
// 2444
```

.filter()

Return a new array with elements from the original array that pass a specific condition.

```
var numOfOranges = [12, 30, 4, 12, 4, 8, 25, 8];
var filterOranges = numOfOranges.filter(function (item) {
  return item > 10;
});
console.log(filterOranges);
// (4) [12, 30, 12, 25]
```

You can also pair `filter()` with `includes()` to check which elements match a string and then produce a new array:

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
var filterOranges = typesOfOranges.filter(function (item) {
  return item.includes("en");
});
console.log(filterOranges);
// (2) ["Clementine", "Valencia"]
```

.map()

Return an array with all elements modified by a callback function, like to multiply the elements.

```
var numOfOranges = [12, 30, 4, 12, 4, 8, 25, 8];
var doubleOranges = numOfOranges.map(function (item) {
  return item * 2;
});
console.log(doubleOranges);
// (8) [24, 60, 8, 24, 8, 16, 50, 16]
```

```
var typesOfOranges = ["Navel", "Clementine", "Valencia", "Bergamot"];
var eat = typesOfOranges.map(function (item){
  return `I want to eat a ${item}!`
}));
console.log(eat);
// (4) ["I want to eat a Navel!", "I want to eat a Clementine!", "I want to eat a
Valencia!", "I want to eat a Bergamot!"]
```

.split()

Split a string into substrings and return these as elements in an array. The type of separator (e.g., a space, no space, or no separator) in the method determines how the string is split.

If the separator is a space, the string is split into array elements where there are spaces:

```
var fruit = "navel orange";
console.log(fruit.split(" "));
// (2) ["navel", "orange"]
```

If the separator is an empty string, the string is split into individual characters:

```
var fruit = "navel orange";
console.log(fruit.split(""));
//(12) ["n", "a", "v", "e", "l", " ", "o", "r", "a", "n", "g", "e"]
```

If there is no separator, the entire string is returned as a single array element:

```
var fruit = "navel orange";
console.log(fruit.split());
// (1) ["navel orange"]
```

.join()

Join all the array elements into a string. If you leave the method empty, the array elements will be separated by a comma.

```
var alphabet = ["a", "b", "c"];

console.log(alphabet.join()); // a,b,c
console.log(alphabet.join("")); // abc
console.log(alphabet.join("-")); // a-b-c
console.log(alphabet.join("?")); // a?b?c
```