***map()* method**

**map()** creates a new array from calling a function for every array element.

* It calls a function once for each element in an array.
* It does not execute the function for empty elements.
* It does not change the original array.
* It returns an array with the results of a functions for each array element.

**Syntax:** *array.map(function(element, index, array){ }, this);*

The example below multiplies every element in the original array by 4. The output will be **[650, 440, 120, 40]**

const numbers = [65, 44, 12, 4];  
const newArr = numbers.map(myFunction)  
  
function myFunction(num) {  
  return num \* 10;  
}

console.log(newArr)

For example, you may have an array of objects that stores **firstName** and **lastName** values as below.

You can use **map()** method to iterate over the array and join the values of **firstName** and **lastName** as below. This would return **['Malcom Reynolds', 'Kaylee Frye', 'Jayne Cobb']**.

const persons = [  
  {firstName : "Malcom", lastName: "Reynolds"},  
  {firstName : "Kaylee", lastName: "Frye"},  
  {firstName : "Jayne", lastName: "Cobb"}  
];  
  
persons.map(getFullName);  
  
function getFullName(element) {  
  return [element.firstName,element.lastName].join(" ");  
}

See another example below, where the **map()** method is used to create another array with just the names.

|  |  |
| --- | --- |
| **Code** | **Result** |
| const items = [  {name: 'Bike', price: 100 },  {name:'TV', price: 200 },  {name: 'Album', price: 10 },  {name: 'Book', price: 5 },  {name: 'Phone', price: 500 },  {name: 'Computer', price: 1000 },  {name: 'Keyboard', price: 25 }  ]  const itemNames = items.map((item) => {  return item.name  })  console.log(itemNames) |  |

***find()* method**

The **find()** method returns the value of the first element that passes a test.

* It executes a function for each array element.
* It returns **undefined** if no elements are found.
* It does not execute the function for empty elements.
* It does not change the original array.

**Syntax**: *array*.find(function(currentValue, index, arr),thisValue)

See the example below, where we it is used to find the “Book” element in the array:

|  |  |
| --- | --- |
| **Code** | **Result** |
| const items = [  {name: 'Bike', price: 100 },  {name:'TV', price: 200 },  {name: 'Album', price: 10 },  {name: 'Book', price: 5 },  {name: 'Phone', price: 500 },  {name: 'Computer', price: 1000 },  {name: 'Keyboard', price: 25 }  ]  const foundItem = items.find((item) => {  return item.name === 'Book'  })  console.log(foundItem) |  |

***forEach()* method**

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

It is not executed for empty elements.

**Syntax:** *array*.forEach(function(currentValue, index, arr), thisValue)

For example, in the example below, for each item in the items array, it will log the price of the element.

|  |  |
| --- | --- |
| **Code** | **Result** |
| const items = [  {name: 'Bike', price: 100 },  {name:'TV', price: 200 },  {name: 'Album', price: 10 },  {name: 'Book', price: 5 },  {name: 'Phone', price: 500 },  {name: 'Computer', price: 1000 },  {name: 'Keyboard', price: 25 }  ]  items.forEach((item) => {  console.log(item.price)  }) |  |

***some()* method**

The **some()** method checks if any array elements pass a test (provided as a callback function).

* executes the callback function once for each array element.
* returns **true** (and stops) if the function returns **true** for one of the array elements.
* returns **false** if the function returns **false** for all of the array elements.
* does not execute the function for empty array elements.
* does not change the original array.

**Syntax:** *array*.some(function(value, index, arr), this)

For example, the code below checks if any of the items has a price below 100, which it has, so it returns **true**.

|  |  |
| --- | --- |
| **Code** | **Result** |
| const items = [  {name: 'Bike', price: 100 },  {name:'TV', price: 200 },  {name: 'Album', price: 10 },  {name: 'Book', price: 5 },  {name: 'Phone', price: 500 },  {name: 'Computer', price: 1000 },  {name: 'Keyboard', price: 25 }  ]  const hasCheapItems = items.some((item) => {  return item.price <= 100  })  console.log(hasCheapItems) |  |

***every()* method**

The **every()** method executes a function for each array element.

* It returns **true** if the function returns **true** for all elements.
* It returns **false** if the function returns **false** for one element.
* It does not execute the function for empty elements.
* It does not change the original array

**Syntax:** *array*.every(function(currentValue, index, arr), thisValue)

For example, not every element passes the test given by the anonymous function, so it will return false.

|  |  |
| --- | --- |
| **Code** | **Result** |
| const items = [  {name: 'Bike', price: 100 },  {name:'TV', price: 200 },  {name: 'Album', price: 10 },  {name: 'Book', price: 5 },  {name: 'Phone', price: 500 },  {name: 'Computer', price: 1000 },  {name: 'Keyboard', price: 25 }  ]  const hasCheapItems = items.every((item) => {  return item.price <= 100  })  console.log(hasCheapItems) |  |

***reduce()* method**

The **reduce()** method executes a reducer function for array element.

* It returns a single value: the function's accumulated result.
* It does not execute the function for empty array elements.
* It does not change the original array.

**Syntax:** array.reduce(*function(total, currentValue, currentIndex, arr), initialValue*)

In the example below, the reduce method uses the callback function to subtract the numbers in the array, starting at the first value (175). So, the return value will be 100 (175-50-25=100).

|  |  |
| --- | --- |
| **Code** | **Result** |
| *const numbers = [175, 50, 25];*  *newArr = numbers.reduce(myFunc);*  *function myFunc(total, num) {*  *return total - num;*  *}*  *console.log(newArr)* |  |

In the example below, it returns the sum of the rounded numbers, which will be 24 (15+2+1+5=24).

|  |  |
| --- | --- |
| **Code** | **Result** |
| const numbers = [15.5, 2.3, 1.1, 4.7];  newArr = numbers.reduce(getSum, 0);  function getSum(total, num) {  return total + Math.round(num);  }  console.log(newArr) |  |

***filter()* method**

The **filter()** method creates a new array filled with elements that pass a test provided by a function.

* It does not execute the function for empty elements.
* It does not change the original array.

**Syntax:** *array*.filter(*function(currentValue, index, arr), thisValue*)

The following array returns an array with the numbers that pass the criteria > 18.

|  |  |
| --- | --- |
| **Code** | **Result** |
| const ages = [32, 33, 16, 40]; const filteredAges = ages.filter(checkAdult);  function checkAdult(age) {   return age >= 18; }  console.log(filteredAges) |  |

See another example, where the **filter()** method is used to create a new array only with the items which the price is below 100.

|  |  |
| --- | --- |
| **Code** | **Result** |
| const items = [  {name: 'Bike', price: 100 },  {name:'TV', price: 200 },  {name: 'Album', price: 10 },  {name: 'Book', price: 5 },  {name: 'Phone', price: 500 },  {name: 'Computer', price: 1000 },  {name: 'Keyboard', price: 25 }  ]  const filteredItems = items.filter((item) => {  return item.price <= 100  })  console.log(items)  console.log(filteredItems) |  |

***includes()* method**

The **includes()** method returns true if an array contains a specified value.

* It returns false if the value is not found.
* It is case sensitive.

**Syntax:** *array*.filter(function(currentValue, index, arr), thisValue)

For example, the code below checks if the number 5 is included in the array. And it is, so it will log **true**.

|  |  |
| --- | --- |
| **Code** | **Result** |
| const fruits = [1,2,3,4,5] const isNumberIncluded = fruits.includes(5);  console.log(isNumberIncluded) |  |