Array Methods

codingwithyash

codingwithyash.com

⊛ Basic Array Operations (methods).

ex → const friends = ['michael', 'steven', 'peter'];

friends.push('Jay');

(i) Push is a function which is used to add an element to the end of an array.

console.log(friends);

output will be → michael, steven, peter, Jay

Push function return the value of length of the new array and we can store the value also like this.

const newlength = friends.push('Jay');

console.log(newlength);

Output will be 4.

(ii) Unshift → It is used to add elements at the beginning of array, and this also returns a value of the length of new array.

friends.unshift('John');

console.log(friends);

output will be → John, michael, steven, peter, Jay

(iii) Pop → This the opposite of Push method, means it will remove the last element of the array.

```
friends.pop();
console.log(friends);
```

output will be → John, michael, steven, peter.

This method return the removed element and we can store it like this.

`friends.pop()` → will remove the last element

```
const popped = friends.Pop();
```

Here we can store that element.

```
console.log(popped);
console.log(friends);
```

→ output → John, michael, steven, Peter

→ output → Jay

[redacted]shift → It is used to remove the first element of an array.

```
friends.[redacted]shift();
console.log(friends)
```

output will be → michael, steven, Peter.

(v) Index of → To find the index of an element

```
console.log(friends.indexOf('Steven'));
```

output will be → 1

If we try to find the index of an element that doesn't exist → the output will be → -1.

ex →

```
console.log(friends.indexOf('Bob');
```

output → -1

(vi) Includes → Returns true, if the element is present in the array and return false if it is not present.

And this checks strictly

→ `friends.push(23);` (23 → number)

friends will become → Michael, Steven, Peter, 23

Now if we check

```
console.log(friends.includes('23'));
```

('23' → string)

So, the output will be false.

Because it is testing with strict equality

(*) Some more array Methods :→

(i) slice method →

```
let arr = ['a', 'b', 'c', 'd', 'e'];

console.log(arr.slice(2));
```

↳ It will not mutate the orginal array but returns an new array

output → ['c', 'd', 'e']

```
console.log(arr.slice(2,4));
```

↳ This one will not be included in the output

output → ['c', 'd']

```
console.log(arr.slice(-2));
```

output → ['d', 'e']

```
console.log(arr.slice(1,-2));
```

output → ['b', 'c']

```
console.log(arr.slice());
```

It will create a copy

output → ['a', 'b', 'c', 'd', 'e']

(ii) Splice Method → It works same as slice but it mutates the original array

```
console.log(arr.splice(2)); → ['c', 'd', 'e']
console.log(arr); → ['a', 'b']
```

It deletes the extracted part from the original array

```
arr.splice(-1);
console.log(arr); → ['a', 'b', 'c', 'd']
arr.splice(1, 2);
```

1 → start, 2 → no. of items to delete

```
console.log(arr); → ['a', 'd']
```

(iii) Reverse Method → It is used to reverse the elements of an array and it also mutate the original array

```
arr = ['a', 'b', 'c', 'd', 'e'];
const arr2 = ['j', 'i', 'h', 'g', 'f'];
console.log(arr2.reversed()); → ['f', 'g', 'h', 'i', 'j']
console.log(arr2); → ['f', 'g', 'h', 'i', 'j']
```

(iv) Concat method → To concatenate two array, It doesnot mutate original arrays.

```
const letters = arr.concat(arr2);
console.log(letters) → ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'
                        'j']
```

(v) Join method → use to join array elements.

```
console.log(letters.join('-'));
a-b-c-d-e-f-g-h-i-j
```


(*) Looping arrays using forEach loop

```
const movements = [200, 450, -400, 3000, -650, -130,
                   70, 1300];

movements.forEach(function(movement)
{
    if(movement>0)
        console.log(`you deposited ${movem
else
console.log(`You withdrew ${math.abs(movement)}`
});
```

If we have to take the index number, we will write it like this.

```
movements.forEach(function(mov, i, arr)
{
  if(mov>0)
console.log('movement ${i+1}:
      you deposited ${mov}');
```

Placement is very important here,
At first current element
At second current index
At third Array

```
else
console.log('movement ${i+1}: you withdrew ${Math.abs(mov)}');
});
```


⊛ Looping Maps and sets using forEach

(1) Maps

```
const currencies = new Map([
    ['USD', 'United States Dollar'],
    ['EUR', 'Euro'],
    ['GBP', 'Pound sterling'],
  ]);

currencies.forEach(function(value, key, map){
  console.log(`${key}: ${value}`);
});
```

Placement matters

(2) Set

```
const currenciesUnique = new Set(['USD', 'GBP',
                        'USD', 'EUR', 'EUR']);

console.log(currenciesUnique);

currenciesUnique.forEach(function(value, _, map){
console.log(`${value}: ${value}`);
});
```

this is underscore which will work as a throwaway variable.

ⓐ Data transformation with map, filter and reduce

map → map returns a new array containing the results of applying an operation on all original array element.

3 1 4 3 2

map → current * 2

6 2 8 6 4

filter → filter returns a new array containing the array elements that passed a specified test condition.

3 1 4 3 2

filter → current > 2

3 4 3 ← filtered array

Reduce → Reduce boils (reduces) all array elements down to one single value. (ex → adding all elements together).

3 1 4 3 2

Reduce → accumulator + current

13 ← Reduced value

⑧ find method

We can use the find method to retrieve one element.

Unlike the filter method, the find method will not return a new array but it will only return the first element in the array that satisfies the condition.

example →

```
const ages = [3, 10, 18, 20];

function checkAge(age)
  {
    return age > 18;
  }

function myFunction() {
document.getElementById("demo").innerHTML
                = ages.find(checkAge)
```

Note: → ① The find method returns undefined if no elements are found.

② The find method does not change the origin array

⊛ findIndex Method

The findIndex method works almost the same way as find method. But the only difference is, findIndex returns the index of the found element and not the element itself.

→ The findIndex() method executes a function for each array element.

→ The findIndex() method returns the index (position) of the first element that passes a test.

→ The findIndex() method returns -1 if no match is found.

→ The findIndex() method does not execute the function for empty array elements.

→ The findIndex() method doesnot change the original array.

Example →

```
const ages = [3, 10, 18, 20];
ages.findIndex(checkAge);
function checkAge(age)
{
  return age > 18;
}
```

(*) Some and Every methods in Javascript.

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

→ The some() method does not execute the function for empty array elements.

→ The some() method does not change the original array

example → const movements = [200, 450, -400, 3000, -650, -130, 70, 1300]

console.log(movements.includes(-130));

[→ output → True, [And Here it is checking equality.]

const anyDeposits = movements.some(mov => mov > 1500);

console.log(anyDeposits)

[→ Output → True, [And Here it is checking for condition]

(*) Every() Method

As the name suggest, every method is pretty similar to the some method, the only difference is that it only returns true if all of the elements in the array satisfy the condition, that we pass in.

⭐ flat and flatMap method.

→ The flat() method concatenates sub-array elements.

→ The flatMap() method maps all array elements and creates a new flat array. It only goes one level deep.

example

```
const arr = [[1,2,3], [4,5,6], 7, 8];
console.log(arr.flat());
```

output → [1, 2, 3, 4, 5, 6, 7, 8]

flat() method by default works on the 1 level deep nested array as show in the above example.

example → 2 level deep nested array

```
const arrDeep = [[[1,2], 3], [4, [5,6]], 7, 8];
console.log(arrDeep.flat());
```

↳ It will give output like this.

[[1,2], 3, 4, [5,6], 7, 8]

but if we want to go 2 level deep, then →

```
console.log(arrDeep.flat(2));
```

↳ [1, 2, 3, 4, 5, 6, 7, 8]

⑧ Sorting Arrays

we can use Javascript built-in method sort() for sorting

example

```
const owners = ['Jonas', 'Zach', 'Adam', 'Martha'];
console.log(owners.sort());
```

↳ ['Adam', 'Jonas', 'Martha', 'Zach']

Note:→ This mutates the original array.

→ The sort method does sorting based on strings.

* const movements = [200, 450, -400, 3000, -650, -130, 70, 1300].

Now we need to sort them using sort method,

If we directly apply sort method on this, then the output will be in the form of string like this →

[-130, -400, -650, 1300, 200, 3000, 450, 70]

So to prevent from this, we will sort them like this :→

```
movements.sort((a, b) => {
  if (a > b)
    return 1;
  if (b > a)
    return -1;
});
console.log(movements)
```

→ Two consecutive numbers)

Note :→
(keep) return < 0, A, B
(switch) return > 0, B, A

-650, -400, -130, 70, 200, 450, 1300, 3000

(*) More ways of creating and filling arrays.

(1) `console.log([1,2,3,4,5,6,7]);`

`console.log(new Array(1,2,3,4,5,6,7));`

```
const x = new Array(7);
console.log(x);
```

↳ output → (7) [empty × 7] → Array of length 7 with 7 empty values.

`x.fill(1,3,5)`

- 1 → value to be filled
- 3 → starting index
- 5 → ending value; It is not included in the array

and if we didn't assign ending value then it will fill till the end.

```
console.log(x);
```

↳ (7) [empty × 3, 1, 1, empty × 2]

- (7) → Array of length 7
- empty × 3 → three empty values
- 1, 1 → [3] [4]
- empty × 2 → two empty value

(2) from() method.

```
const y = Array.from({length: 7}, () => 1);

console.log(y);
```

[1, 1, 1, 1, 1, 1, 1]

```
const z = Array.from({length: 7}, (curr, i) => i+1);

console.log(z)
```

Thank You !!!

codingwithyash

codingwithyash.com