Arrays in JavaScript

JavaScript Arrays Objects are used to collect multiple datatypes in a single variable. **Arrays** are list-type objects under Reference datatypes in JavaScript. They are build-in objects to store multiple data.

Arrays are indexed collections in javascript.

In JavaScript, **Arrays** can store multiple data in a single variable. But in *strictly typed* programming languages, Arrays can store only single datatype. This means, in JavaScript, arrays can store any type of data. A **JavaScript Array** can store both string, number or any other datatype in same array.

JavaScript Arrays

How to declare Array in JavaScript

Array in JavaScript can be declared using brackets[] or *new Array()* constructor. Both *[]* and *new Array()* works same. **new Array()** is constructor form, and brackets [] is literal form of **Array**.

Array Literal

```
let month=[];            // blank Array
```

Array Constructor

```
let month=new Array();        // blank Array
```

All arrays declared using `[]` or `new Array()` are instance of **Array Class or Function**. The `typeof Array` is function in JavaScript.

`[]==[]` or `new Array()==new Array()` is always false in javascript.

Check Array datatype

Arrays are Reference data types. typeof operator will return "object". To check datatype of a arrays, use Array.isArray() function or method. **Array.isArray()** function will determine whether the passed value is an Array or not.

```
let x=[];
typeof x;            // return "object"
Array.isArray(x);    // return true
```

Traversal values of array

To check element inside array, we can call array followed by index value in brackets. To find first element, call array[0]. This index notation starts from 0 to array.length-1.

```
let days=["sun","mon","tues"];

    days[0];        return "sun";
    days[1];        return "mon";
    days[2];        return "tues";
    days[3];        return undefined;
```

array.length

array.length property is used to check total numbers of elements inside array. Default length is zero, for an **empty array**. For a non empty Array, array.length is positive number.

```
let x=[];
let y=["jan","feb","mar"];

x.length;        return 0;
y.length;        return 3;
```

In JavaScript, **Array length is mutable**. This means, we can manually change length of an array. If default length is 3, and we assign array.length=5, new length of array will be 5 with two *undefined*.

```
let days=["sun","mon","tues"];
days.length;   //return 3

days.length=5;
days.length;   //return 5

days           //return ["sun","mon","tues",,] 2 empty

days.length=6;
days.length;   //return 6

days           //return ["sun","mon","tues",,,] 3 empty
```

Array Methods

In JavaScript, **Arrays** are used to store multiple data inside single variable. Arrays have properties are methods. Array.length is Array property. Now we will discuss **array methods**.

sort()

array.sort() will sort the order an array. This will change Alphabetical order or an array. **array.sort()** will also change order of an array permanently.

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];

days.sort();
days
// return ["fri", "mon", "sat", "sun", "thurs", "tues", "wed"]
```

sort method will also sort numeric array in alphabetical order.

```
let i=[1,2,3,10,20];
i.sort();
i;  return [1,10,2,20,3];
```

Sort numeric array

To sort numeric array, pass a callback function in sort method.

```
let i=[1,2,3,10,20];
function srt(a,b){ return a-b };
i.sort(srt);
console.log(i);  //return [1,2,3,10,20];
```

reverse()

array.reverse() method is used to reverse an array. this will bring last element to first and first element to last.

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];
days.reverse();
days;
    // ["sat", "fri", "thurs", "wed", "tues", "mon", "sun"]
```

indexOf()

array.indexOf() method is used to check index of first element in array.

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];

days.indexOf("sun");         // return 0
days.indexOf("sat");         // return 6
days.indexOf("SUN");         // return -1
```

use array.lastIndexOf() to check last index of an element in array.

shift

array.shift() method is used to remove first element from an array. This will reduce array length by 1 as first element is removed.

```
let days=["sun","mon","tues"];

days.shift();        // return "sun"
days;               //return ["mon","tues"]
```

unshift

array.unshift() method is used to add first element in an array. This will increase array length by 1 as one element is added on zero index.

```
let days=["sun","mon","tues"];

days.unshift("DAYS");     // return 4
days;                     //return ["DAYS","sun","mon","tues"]
```

pop

array.pop() method is used to remove last element from an array. This will also decrease array length by 1 as last element is removed .

```
let days=["sun","mon","tues"];

days.pop();     // return "tues"
days;                     //return ["sun","mon"]
```

push

array.push() method is used to add an element in array ar last position. This will also increase array length by 1 as one element is added.

```
let days=["sun","mon","tues"];

days.push("wed");     // return 4
days;                      //return ["sun","mon","tues","wed"]
```

splice

array.splice() method is used to add or remove n number of elements from an array at any position. This will also increase or decrease array length by n as n elements are added or removed.

```
let days=["sun","mon","tues"];

days.splice(0,1);  // remove 1 element from 0 index
days               // return  ["mon","tues"];

days.splice(1,1);  // remove 1 element from 1 index
days               // return  ["sun","tues"];

days.splice(0,2);  // remove 2 element from 0 index
days               // return  ["tues"];

days.splice(0,0,"days"); // add 1 element at 0 index
days                     // return  ["days","sun","mon","tues"];

days.splice(0,1,"days"); // add 1 element by removing element at 0 index
days                     // return  ["days","mon","tues"];
```

pop, push, shift, unshift and splice methods can manipulate array data.

slice

array.slice() method is used to slice a single element from array. This will not change actual array. **slice** can return single element if parameter passed is single. For two parameter(x,y), slice can return y-x elements.

```
let days=["sun","mon","tues"];

days.slice(0);        return ["sun","mon","tues"];
days.slice(1);        return [mon","tues"];
days.slice(2);        return ["tues"];
days.slice(1,3);      return ["mon","tues"];
```

join

array.join() method is used to **convert an array to string** with all elements inside array, separated by commas. To change default separator from commas to any other character by passing that inside parenthesis.

```
let days=["sun","mon","tues"];

days.join();        return "sun,mon,tues";
days.join(":");     return "sun:mon:tues";
days.join("-");     return "sun-mon-tues";
```

concat

array.concat() method is used to **merge or concat an array to another array**. After join, we will get a new array.

```
let days1=["sun","mon","tues"];
let days2=["wed","thurs","fri","sat"];

let days=days1.concat(days2);

days;  // return ["sun","mon","tues","wed","thurs","fri","sat"]
```

concat using es6 spread operator ...

In ES6, we can also use **spread operator**, i.e `...` to concat array elements.

```
let days1=["sun","mon","tues"];
let days2=["wed","thurs","fri","sat"];

let days=[...days1,...days2];

days;  // return ["sun","mon","tues","wed","thurs","fri","sat"]
```

flat method

Array.flat() method create a new array from existing nested arrays upto specific depth.

```
    let arr1=[1,2,3,[4,5]];
    let arr2=arr1.flat();
    console.log(arr2);                                    // [1,2,3,4,5]
```

flat depth

we can optionally pass **depth parameter in flat** method to specify how deep the nested array should flattened.

```
let arr1=[1,2,3,[[4,5]]];

let arr2=arr1.flat();
console.log(arr2);                                          // [1,2,3,[4,5]]

let arr3=arr1.flat(2);
console.log(arr3);                                          // [1,2,3,4,5]
```

Advance Array Methods

Here are some **advanced array methods** used in javascript arrays.

Map

Array.map() method create a new array by using an existing array and output of callback function in **map method**. An callback function is required in map method with parameter to iterate each element.

[2,4,6]

```
let arr=[1,2,3];
let arr2=arr.map(function(i){ return i+i});

console.log(arr2);
```

[1,4,9,16]

```
let arr=[1,2,3,4];
let arr2=arr.map(function(i){ return i*i});

console.log(arr2);
```

Reduce

Array.reduce() method reduce the array into a single value by passing callback function. For example, we can use **reduce method** to sum all elements in an array.

Sum using reduce method

10

```
let arr=[1,2,3,4];
let red=arr.reduce((i,j)=>{ return i+j});

console.log(red);
```

Multiply using reduce method

24

```
let arr=[1,2,3,4];
let red=arr.reduce((i,j)=>{ return i*j});

console.log(red);
```

Filter

Array.filter method creates a new arrays with filtered data from callback.

Filter strings by length

["tues", "thurs"]

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];
let filter=days.filter( i => i.length>=4 );
```

Filter numbers greater than

[5,7,9]

```
[1,3,5,7,9].filter(i=> i>3 );
```

Filter Even numbers

[10,4,22]

```
[10,3,4,9,22].filter(i=> i%2==0 );
```

find

Array.find method returns first match element in given array based on condition. This method will iterate over array and return first matched element.

5

```
[1,3,5,7,9].find(i => i>=5 );
```

```

```

1

```
[1,13,5,17,9].find(i => i<5 );
```

findIndex

Array.findIndex method returns index of first match element in given array based on condition. This method will iterate over array and return first matched element.

2

```
[1,3,5,7,9].findIndex(i => i>=5 );
```

3

```
[1,13,5,17,9].findIndex(i => i>=15 );
```

fill

Array.fill method fill array elements with a static value. We can also set start index and end index using 2nd and 3rd parameters.

[0,0,0,0,0]

```
[1,3,5,7,9].fill( 0 );
```

[1,3,0,0,0]

```
[1,3,5,7,9].fill( 0,2 );
```

[1,3,0,0,9]

```
[1,3,5,7,9].fill( 0,2,4 );
```

some

Array.some method test whether at least one element in given array fulfill test condition in callback. The possible output are `true` or `false`.

false

```
    [1,3,5,7,9].some( i=> i>=10 );
```

false

```
    [1,3,5,7,9].some( i=> i%2==0 );
```

```

```

true

```
[1,3,5,7,9].some( i=> i%5==0 );
```

every

Array.every method test whether all elements in given array fulfill test condition in callback. This will also returns `true` or `false`.

true

```
[1,3,5,7,9].every( i=> i<10 );
```

true

```
[1,3,5,7,9].every( i=> i%2!=0 );
```

false

```
[1,3,5,7,9].every( i=> i%5==0 );
```

Iterate Array

There are various ways to **iterate over an array** in JavaScript. Here is list of arrays method and loops available for array data iteration.

1. forEach()
2. For in Loop
3. For of Loop

forEach

forEach method is used to get all elements in array one by one without using loop. A callback function is added as a parameter in **forEach**.

Get all elements from array using forEach

sun

mon

tues

wed

thurs

fri

sat

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];

days.forEach(function(i){console.log(i)});
```

Get all elements with index from array using forEach

sun 0

mon 1

tues 2

wed 3

thurs 4

fri 5

sat 6

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];

days.forEach(function(i,j){console.log(i,j)});
```

For in loop Array

For in loop is also used for arrays to get each element one by one. The advantage of **for in** over simple for loop is that **for in** loop require variable initialization and array name. Increment is not required. See example

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];

for( let i in days){
    console.log(i);            // index of element
    console.log(days[i]);      // element
}
```

For of Loop Array

forEach method is best for traversal of arrays as the value of i in callback function is each element. **For in loop** is also good, but to get elements, we have to use array[i]. Now we have **forEach** features in for loop using **For of**.

For of loop in ES6 returns array element, instead of index.

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];

for( let i of days){
    console.log(i);            // element
}
```

Bind array data is select dropdown

Day:

--Choose Day--

```
let days=["sun","mon","tues","wed","thurs","fri","sat"];

for( let i in days){
    document.querySelector("select").innerHTML+=`<option> ${days[i]} </option>`;
}
```

Multidimensional Array

Multidimensional Array means an **Array inside another array** or an **array of arrays**. We can store n numbers of **arrays inside array**.

```
let students=[["g1","g2","g3"],["b1","b2","b3","b4"]];

students.length;              // 2 arrays in array
students[0];                  // first array in students
students[1];                  // second array in students
students[0].length             // 3
students[1].length             // 4
students[0][0];               // "g1"
```

```
students[0][1];                 // "g2"
students[0][2];                 // "g3"
students[1][0];                 // "b1"
students[1][1];                 // "b2"
students[1][2];                 // "b3"
students[1][3];                 // "b4"
```

Bind 2d array in table

g1	g2	g3	g4
b1	b2	b3	b4

```
let data=[["g1","g2","g3","g4"],["b1","b2","b3","b4"]];

for( let i in data){
    let tr = document.createElement("tr");
    for( let j in data[i]){
        tr.innerHTML+="<td>"+ data[i][j]+"</td>";
    }
    document.querySelector("table").appendChild(tr);
}
```