Arrays and Objects

Created	@October 3, 2022 8:00 PM
Class	
Type	
Materials	
Reviewed	☐

Objects

In JS, objects are a special data structure, that can store <Key, Value> pairs. A lot of people think everything in JS is an object, this statement is 101% wrong.

ECMAScript® 2019 Language Specification
https://262.ecma-international.org/10.0/#sec-ecmascript-data-types-and-values

Note: There is a historical mistake in JS, due to which when you try to get the type of `null` you get `object`. They don't rectify it because now there is huge code base of JS based applications that might break due to this change, and hence we move forward with this problem.

In other languages also, we have dedicated data structures that can store key value pairs, like in python we have `dictionary`, ruby has `hashes` , cpp has got `unordered_map` etc.

```
let x = {name: "Sanket Singh", profession: "Software Engineer"}
console.log(x);
```

The above code snippet gives a basic example of a JS object.

How to add values in JS objects ?

```
x["city"] = "Bengaluru"; // example 1

console.log(x);

x.company = "Google"; // example 2

console.log(x);
```

So here, `x` is the variable name which stores an object. So if we want to add a key value pair there are 2 ways

- We can use dot operator, using which we can say `<variable>.<key> = <value>` this is demonstrated in example 2.
- The other way to add a key value pair is using `[]` , we can say `<variable>["<key name as string>"] = <value>` demonstrated in example 1.

How to delete key value pairs in JS objects ?

```
delete x.company;

console.log(x);
```

we can use the `delete` keyword, and mention the object name along with the key to be deleted.

Iterables in JS

An iterable is an object which can be looped over or iterated over with the help of a loop.

Spread operator

So the `...` is called as the spread operator. We can use the spread operator to create copy of the values of any iterable. By default, it doesn't support objects, but what we can do is, we can apply the spread operator on an object, to unpack it's values and then immediately pack those values back in other object

```
let x = {name: "Sanket", company: "Google"};
// let x = [1,2,3,4];
console.log({...x});
```

So what is the use of it ?

We can use this spread operator to merge two or more objects into one.

```
let x = {name: "Sanket", company: "Google"};
let y = {professsion: "Software engineer"}
// let x = [1,2,3,4];
console.log({...x, ...y});
```

How to iterate on an object ?

We can use `for in` loop to loop over the keys.

```
x = {name: "Sanket", company: "Linkedin"}
for(let element in x) {
  console.log(element, x[element]);
}
```

using `Object.keys` we can get an array of keys.

```
x = {name: "Sanket", company: "Linkedin"}
console.log(Object.keys(x));
```

using `Object.values` we can get an array of all values

```
x = {name: "Sanket", company: "Linkedin"}
console.log(Object.values(x));
```

using `Object.entries` we can get an array of key value pairs

```
x = {name: "Sanket", company: "Linkedin"}
console.log(Object.entries(x));
```

Object.freeze

This function, actually makes the object non configurable towards any incoming changes.

```
x = {name: "Sanket", company: "Linkedin"}
Object.freeze(x);
x.name = "JD";
x.salary = 100000000;
console.log(x);
// no changes in the object x will occur
```

In strict mode, this will throw error.

Object.freeze() - JavaScript | MDN

The Object.freeze() method freezes an object. Freezing an object prevents extensions and makes existing properties non-writable and non-configurable. A frozen object can no longer be changed&colon;

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze

Arrays

Arrays are special objects, which stores data in linear fashion, in contiguous memory locations.

```
let x = [1,2,3,4];
console.log(typeof x); // object
```

Here , we can say that indexes of an array act as a key and the values at those indexes act as values. JS arrays are 0 based indexed. JS doesn't support negative indexes.

How to access an element in an array ?

We can write `<name of array>[<index>]`

```
x = [1,2,3,4]
console.log(x[1]); // 2
```

Important method on arrays

arrays.map

This function takes an input argument which is another function only. So the `map` function takes every element of the array, and passes it to the function given in the argument. It doesn't change the original array, instead returns a new array where elements of the new array are the result of the return values passed in the argument function.

```
let x = [1,2,3,4];
y = x.map(function (element) { return element * 2; });
console.log(x); // Array(4) [ 1, 2, 3, 4 ]
console.log(y); // Array(4) [ 2, 4, 6, 8 ]
```

arrays.filter

This method also takes input argument as a function, and returns a new array. What are the elements of the new array ? So the argument function of filter takes every element of the given array one by one, and there is conditional return statement in the filter function. So all those element which satisfies the condition, gets filtered out in a new array.

```
let x = [1,2,3,4];
y = x.filter(function (element) { return element % 2 == 0; });
console.log(x); // Array(4) [ 1, 2, 3, 4 ]
console.log(y); // Array(4) [ 2, 4 ]
```

```
let x = [-1, 0, 1, 2, 3, 4];
y = x.filter(function (element) { return element; });
```

```
console.log(x); // Array(6) [ -1, 0, 1, 2, 3, 4 ]
console.log(y); // Array(6) [ -1, 1, 2, 3, 4 ]
```

So in the above example, the argument function is not returning a conditional value. So how filter function will filter the element ?

So here, what actually happens is, filter function at last converts the result returned into a boolean. If it is already a boolean no problem, if it is not then for `0, false, -0, null, undefined, "", NaN` these values will return false (doesn't satisfy condition), else everything will return true (satisfies condition).

arrays.includes // O(n)

It checks whether element exists in the given array or not ?

```
let x = [-1, 0, 1, 2, 3, 4];
y = x.includes(2);
console.log(y); // true
```

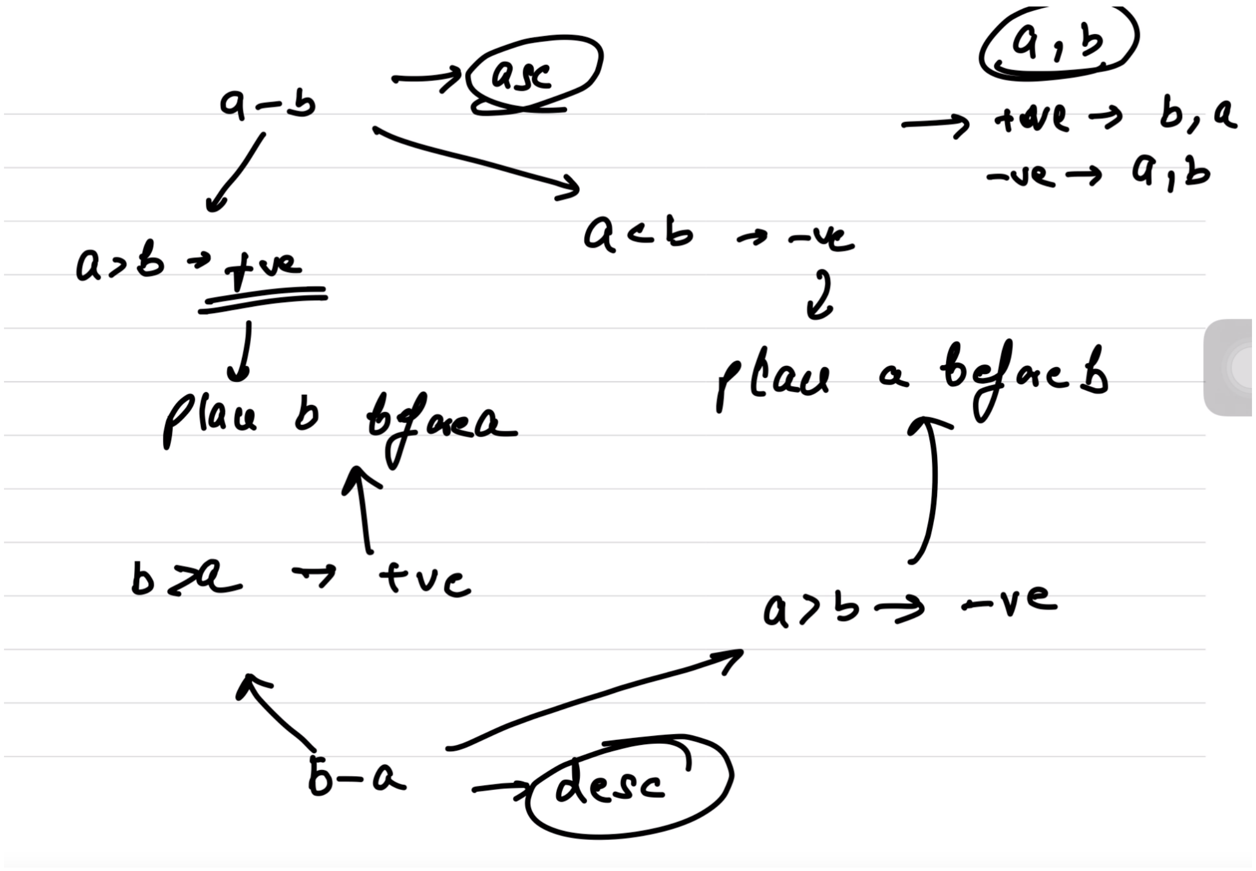
arrays.sort

By default, sort function arranges the array in dictionary order / lexicographical order.

```
let x = [-1, 6, 1, 2, 0, 32, 19];
x.sort();
console.log(x);// [ -1, 0, 1, 19, 2, 32, 6 ] // A AB B
```

So the sort function takes an optional argument which is a function. If you leave it then it will sort the array in dictionary order otherwise if you pass a function, it will use that function to actually do the comparison.

```
let x = [-1, 6, 1, 2, 0, 32, 19];
x.sort(function (a, b) { return a - b;});
console.log(x); // [ -1, 0, 1, 2, 6, 19, 32 ]
```



arrays.push

it can add a value to the last of the array, insert at end of the array. Makes changes in the same array

arrays.pop

It can remove the last element of the array. Makes changes in the same array

```
x = [1,2,3,4,5];
x.push(10);
console.log(x); // [1,2,3,4,5,10]
x.pop();
console.log(x); // [1,2,3,4,5]
```

arrays.unshift

This can insert element to the start of the array

arrays.shift

This can remove the element from the start of the array.

```
x = [1,2,3,4,5];
x.unshift(10);
console.log(x); // [10,1,2,3,4,5]
x.shift();
console.log(x); // [1,2,3,4,5]
```

arrays.reverse

This can reverse the given array.

arrays.indexOf

This function takes an argument, and tries to search for the argument. If it finds it , then returns the index of the first occurrence of the element, otherwise returns -1.

arrays.every

This function takes an argument as a function only. And then returns true of false based on the fact that whether all the elements passed the condition of the argument function or not ? If any one of the element fails, then also it will return false.

```
x = [2,4,6];
console.log(x.every(function (e) { return e%2 == 0})); // true

x = [11,2,4,6];
console.log(x.every(function (e) { return e%2 == 0})); // false
```

arrays.some

This check whether at-least one element satisfies the condition of the argument function or not ?