

JavaScript – old JS, ES5, ES6, and ES7 features needed in e.g. React development and seen in many React/Redux/Material UI/Node backend examples.

The list of the ES/(TS) features needed in React development. Some are even older than ES5, but tricky. Focus is on features that are 1. tricky, 2. risky, 3. confusing, 4. treacherous, 5. difficult, 6. error-prone or many of those.

See the Mozilla Developer Network links for all of these!

- **let** – block-scoped variable (Until ES6 we only had 'var' with only two possible scopes: function and global, though implicit global vars and var hoisting cause some risks if not coding well)
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>
- **const** - block-scoped constant (the first immediate value needs to be assigned right away and will be constant, e.g. the object reference. But the _contents_ of that object and so on are not protected by const!).
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const>
- **pass-by-value** (JavaScript and e.g. Java have only pass-by-value) vs. **pass-by-reference** (E.g. C/C++/C# have also this). For beginners it must be confusing to pass values that are references and still understand it as pass-by-value!
When we pass the value of the reference value, a reference, it's still pass-by-value. Only if we would be able to pass a reference to that original reference variable itself, it would be pass-by-reference! (Then the function could change the value of the original variable, which is only rarely desirable)
- **shallow copy** (first layer of objects is duplicated as separate objects, but after that the references refer to original second layer objects = not independent copy) vs. **deep copy** (all objects in the, even deeper object structure are duplicated as separate objects and the original objects are safely separate)
- **arrow functions** (shorter syntax, implicit return, reference 'this' auto-bound to outer scope, 2 more)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
- **.map** method/function
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- **.forEach** function for many kind of collections
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/forEach
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/forEach
- **.filter** method
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter
- **.reduce** method
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce
- ((ES6 **class** syntax <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class>))
- ((ES6 class **inheritance** syntax
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/extends>))

- (A **new way of defining methods** (Methods: object-attached functions, object's function members)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Method_definitions#Description)
- **template literals** and **placeholders** (with backticks ` and `\${ }` to get rid of this kind of String concatenation clumsiness: "Hello"+name+"!")
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals
- **spread operator** (spread notation/spread syntax) to make 'a bit deeper shallow copy' of an object, instead of the 'totally shallow copy'. Copying goes **one level deep** = the properties/items of the original and copy object are separate. (But those separate properties/items may then contain references to exactly same objects)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator
- ES6 **export** and **import** from a **module** to another (default export or named export)
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export>
 - So after ES 2015 = ES6 this version has been spreading wider in JS:
 - in original.js **export default** someObj; // default export (one per module)
 - in file using.js **import** myObj **from** './original'; // default import and naming 'myObj'
 - in original.js **export** someObj; // named export (multiple items possible)
 - in file using.js **import** {someObj **as** myObj} **from** './original'; // named import (and rename)
 - (It replaced the older the CommonJS way: <https://en.wikipedia.org/wiki/CommonJS>)
 - (in original.js **module.exports** = someObject; // exposing someObject as/from module)
 - (in file using.js var copyOfSomeObject = **require**('./original.js'); // getting an instance of it)
- extra **trailing comma** was allowed at the end of lists already in old JS. ES5 added it to object literals and ES8 to functions. [1,2,3,] {name:"Joe",yob:1986,} foo(2,3,);
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Trailing_commas
- **Property accessor** used so that its name is not hard-coded string, but comes from a variable:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer Scroll down to "Computed property names".

```
this.setState({[event.target.name]: event.target.value});
```

compare to this: this.setState({firstName: event.target.value});
when the event target's name was string "firstName". Note: same feature as in our **{[a]:a,[b]:b}** example

- **short notation object literals** of this kind: { a } which means same as { a : a }
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer#New_notations_in_ECMAScript_2015

Curiosity: In React JSX **{{a}}** means first going to JS mode using the outer { } and then having that shortened {a} object literal inside

- **Destructuring assignment.** Destructuring assigned object or array values into separate local variables
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

- **optional chaining** (Should we access the member property or function, or use the value "undefined" instead). Happens if the parent item on the left is either: missing, explicitly with value undefined, or null:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining

OLD JS, But still tricky and good to know

- function **parameter default values**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters
- **leaving arguments out is only allowed at the end of a function argument** list while calling a function
That's why we need to write e.g. `(_, index) => index%2==0` where we are marking the skipped parameter with dummy name `_`. That is counted as a parameter, but not needed/used. We need to write the `_` as otherwise index would not be the second parameter like it needs to be. Similar use:
`(_ => whatever_code_here)`
- **falsy values.** Anything that will be considered false while e.g. given to if condition: if(a) (Almost in every exam!)
<https://developer.mozilla.org/en-US/docs/Glossary/Falsy> (You could remember 3-9 from Scrum team size)
<https://developer.mozilla.org/en-US/docs/Glossary/Truthy>
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Equality_\(\)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Equality_())
- **IIFE, SIAF, SEAF** <https://developer.mozilla.org/en-US/docs/Glossary/IIFE> Learn the first example(s) here:
<https://developer.mozilla.org/en-US/docs/Glossary/IIFE#Examples>
- (Smaller curiosity) JavaScript doesn't allow **identifiers starting with number**. But what if you get the JSON text `{"123": "Yeah"}` and parse it as an JavaScript object?

```
var a = JSON.parse('{"123": "Yeah"}');
```



```
console.log(a.123);    // Error, unexpected number
```



```
console.log(a."123");  // Error, unexpected String
```



```
console.log(a["123"]); // ok, prints: Yeah
```



```
console.log(a[123]);   // ok, prints: Yeah
```


https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer#Accessing_properties
- **Difference between JavaScript Object literals (=JavaScript code) and JSON (=Text, String in JS, thus not JS):**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer#Object_literal_notation_vs_JSON So JSON is not JavaScript, but is text that is compatible with JS object serialization.

+SELECTED BASICS OF TYPESCRIPT

https://github.com/valju/JS_ES_Features/blob/master/TS_basics/TS_in_Siba_project.md

Each ES (or TS) feature you will now learn gets you closer to max points in exam! On the otherhand you can make some 3-4 mistakes and still give 100% result for the aggregate. Thus every 10-15 minutes counts. Every play with code in MDN.

***** **END** *****