JavaScript – (old ES/JS,) ES5, ES6, and ES7 features needed in React development and seen in many React examples.

The list of the ES features needed in React development. Some are older than ES6 though…

See the Mozilla Developer Network for all of these!

- **let** - block scoped variable (Until ES6 we only had 'var' with only two possible scopes: function and global) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let
- **const** - block scoped constant (the first immediate value needs to be given 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
- **arrow functions** (shorter syntax, implicit return, reference 'this' auto-bound to outer scope) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
- **.map** 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
- 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
- **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 'deeper copy' of an object, instead of the 'totally shallow copy'. Copying goes **one level deep** = the properties of the original and copy object are separate. (But those separate properties may contain references to same objects) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator
- ES6 **export** and **import** from **module** to another (default or named) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export
- ES6 **promises** (promise1.then(function2) ….) Easier to read handling of asynchronous function calls and their callbacks. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- extra **trailing comma(s)** allowed at the end of ES6(?) lists etc. e.g. [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});`

if the event's target's name was string "firstName".

- OLD JS: function **parameter default values**
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters
- OLD JS: **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) => <li key={index}>index 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)
- OLD JS: **falsy values** and them in type-coerced comparisons with only two equals signs ==
  https://developer.mozilla.org/en-US/docs/Glossary/Falsy
  https://developer.mozilla.org/en-US/docs/Glossary/Truthy
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Equality_()
- **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

  In React JSX {{a}} means first going to JS mode using the outer { } and then having that shortened {a} object literal inside
- **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
- JavaScript doesn't allow **identifiers starting with number**. But what if you get JSON {"342":"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

Not in the 2018-12-10 exam: (Some of these **<u>just because</u>** they were **<u>not included yet</u>, possibly will be <u>in future exams</u>**)

OUT OF SCOPE: The items below in this list:

- New way of defining methods (Object-attached functions, object's function members) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Method_definitions#Description
- **async functions** with an implicit Promise and possible await inside: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
- JavaScript **closures**
- **Difference between** JavaScript **Object literals** and **JSON**: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer#Object_literal_notation_vs_JSON

Most likely **not in future exams either** even if useful

- OUT OF SCOPE l: Intl API.
-

Going to the other exams:

- Ready-made React UI Components, like the react-table package. After installing package check where everything went.