JavaScript ES 6 review

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Let and Const
- Arrow Functions
- Exports and Imports
- Classes
  - Constructor
  - Extend
- Spread Operator
- Destructuring
- Array Functions

Let & Const

- **Best practice** is always start with const. If you need to change its value, then make it let.

- When using Let and Const, the variable is only available in the **block** it's defined in.
- **Const** is a signal that the identifier won't be reassigned.
- **Let** is a signal that the variable may be reassigned, such as a counter in a loop.

Arrow Functions

- Were introduced in ES6.
- Allow us to write shorter function syntax.
- An arrow function expression is a syntactically compact alternative to a regular function expression, although without its own bindings to the **this**, arguments, super, or new.target keywords.
- Always use arrow functions in React.
- “You won’t introduce any bugs by using too many arrow functions. You probably will introduce bugs but not using enough.” reactarmory.com.

Arrow Functions Syntax

```
(param1, param2) => { statements } ;

(param1, param2) => expression; // An expression is any valid
unit of code that resolves to a value.

// Parentheses are optional when there's only one parameter
name:

(singleParam) => { statements }

singleParam => { statements }

// The parameter list for a function with no parameters
// should be written with a pair of parentheses.

() => { statements }
```

Arrow Functions Examples

If the function has only one statement, and the statement **returns** a value, you can remove the brackets and the return keyword.

```
const sayHello = (name) => {
  return 'Hello ' + name;
}
```

```
const sayHello = name => {
  return 'Hello ' + name;
}
```

```
const sayHello = name => 'Hello ' + name;
```

this in Arrow Functions

- There are **no binding** of this.
- In regular functions, the **this** keyword represents the object that called the function, which could be the window, the document, a button or whatever.
- this keyword in arrow functions always represents the object that defined the arrow function.
- Once you've passed a function as a callback, you have no idea how it will be called, and thus no idea what **this** will be. But arrow functions solves this issue and your code is more consistent and has less bugs.

Exports

The export statement is used when creating JavaScript modules to export live bindings to functions, objects, or primitive values from the module so they can be used by other programs with the import statement

Exports

There are two types of exports:

1. Named Exports (Zero or more exports per module)
2. Default Exports (One per module)

Named Exports

- Named exports are useful to export several values.
- During the import, it is mandatory to use the same name of the corresponding object.

Named Exports

```
export let fname, lname;
export let fname = 'umur', lname = 'inan';
export function functionName(){...}
export class ClassName {...}
// Export list
export { fname, lname };
// Renaming exports
export { fname as firstname, lname as lastname };
```

Default Exports

```
export default expression;

export default function (…) { … }

export default function name1(…) { … }

export { name1 as default };
```

Import

import statement is used to import read only live bindings which are exported by another module.

Imports

```
import defaultExport from "module-name";
import { export1 } from "module-name";
import { export1 as alias1 } from "module-name";
import { export1 , export2 } from "module-name";
```

Exports and Imports

```
//student.js
const student = {
  name : 'bob'
}
export default student;
```

```
//helper.js
export const minutesInHour = 60;
export const sayHi = () =>
'Hello';
```

```
//app.js
import student from './student.js';
import stu from './student.js'
import {minutesInHour} from './helper.js';
import {sayHi} from './helper.js';
```

Class

- Classes are a template for creating objects. They encapsulate data with code to work on that data.
- Classes in JS are built on prototypes. Prototypes are the mechanism by which JavaScript objects inherit features from one another.
- Classes are in fact "special functions", and just as you can define a function.
- An important difference between function declarations and class declarations is that function declarations are hoisted and class declarations are not.

Class Example

```
class Student extends Person {
 constructor(name){
   super();
   this.name = name;
 }
 sayHi = () => 'Hi ' + this.name;
}
const student1 = new Student('Bob');
console.log(student1.sayHi());
```

Constructor

- The constructor method is a special method for creating and initializing an object created with a class.
- There can only be one special method with the name "constructor" in a class.
- A constructor can use the super keyword to call the constructor of the super class.

Extend

- The extends keyword is used in class declarations or class expressions to create a class as a child of another class.
- If there is a constructor present in the subclass, it needs to first call super() before using "this".

Spread Operator

Used to split up array elements or object properties. So we can expand, copy an array, or clone an object.

```
function sum(x, y, z) { return x + y + z; }
const numbers = [1, 2, 3];
console.log(sum(...numbers));

let obj1 = { foo: 'bar', x: 42 };
let clonedObj = { ...obj1 };
let mergedObj = { ...obj1, ...obj2 };
```

Destructuring

- Extract array elements or object properties into variables.
- Spread operator takes all the elements or all the properties whereas destructuring pulls out single element or single property to variables.

Destructuring Example

```
let [a,b] = ['Hello','World'];
console.log(a); // Hello
console.log(b); // World
let student = {
   name: 'Bob',
   age: 20
 };
 let {name} = student;
 console.log(name); // Bob
 let {age} = student;
 console.log(age); // 20
```

Array Functions

map, find, findIndex, filter, reduce, slice (end is not included), splice.

map - creates a new array populated with the results of calling a provided function on every element in the calling array.

```
let numbers = [1, 2, 3];
let doubleNumbers =
numbers.map((item, index) => {
   return item * 2;
});
```

Array Functions - Filter

filter(start, deleteCount, item1) - creates a new array with all elements that pass the test implemented by the provided function.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destructi
on', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]
```

Array Functions - Find

find((element) => { ... }) - returns the value of **the first element** in the provided array that satisfies the provided testing function.

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
// expected output: 12
```