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
- Call/apply/bind
- 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.
- Use const for JSON objects.
- 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.

What will it print?

```
(function timer() {
  for (var i=0; i<=5; i++) {
    setTimeout(function clog() {console.log(i)}, i*1000);
  }
})();
```

Solution I

```
(function timer() {
  for (let i=0; i<=5; i++) {
    setTimeout(function clog() {console.log(i)}, i*1000);
  }
})();
```

Solution II

Alternatively, developers used to fix this bug before ES6 using immediately called functions. Immediately called functions also give data privacy.

```
(function timer() {
  for (var j = 0; j <= 5; j++) {
    (function () {
      var i = j;
      setTimeout(function clog() { console.log(i); }, i * 1000);
    }());
  }
})();
```

Arrow Functions

- Were introduced in ES6.
- Allow us to write shorter function syntax.
- An arrow function is not just a syntactic sugar. There are differences between a regular function and arrow function. For example, the **“this”** keyword.
- Always use arrow functions in React class-based components.

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;
```

The “this” keyword

This is the object that owns the functions in JavaScript.

```
const test = {
  prop: 42,
  func: function() {
    return this.prop;
  },
};
console.log(test.func());
```

A function's this keyword behaves a little differently in JavaScript compared to other languages. ES5 introduced the **bind()** method to set the value of a function's this regardless of how it's called.

The “this” in Arrow Functions

- In regular functions, the **this** keyword represents the object that called the function, which could be the window, the document, a button or whatever.
- The “this” keyword in arrow functions always represents the object that defined the arrow function. Because it implicitly bind the function to the parent’s this. It is **consistent** and predictable. Hence, less bugs.

call/bind/apply methods

- These methods allow you specify the “this”.
- One of the use case is to borrow a function from another object and carry a function. The first parameter is the “this” object and the rest is the other parameters to the function.
- Call – Takes parameters one by one
- Apply – Takes parameters as an array
- Bind – Returns another function with the giving object as the “this”.

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

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 as tellHi} from './helper.js';
```

Require vs import

- Require for importing modules in NodeJS is built on top of the "Common JS". The Common JS is for organizing code in JS in a modular fashion. Later, ECMAScript standardized JS modularity using import/export.
- Require can be used inside if/else conditions just like an expression or a variable where import is only used on top of the JS file.
- Require imports module synchronously whereas the "import" can import asynchronously.

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.

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.

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());
```

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

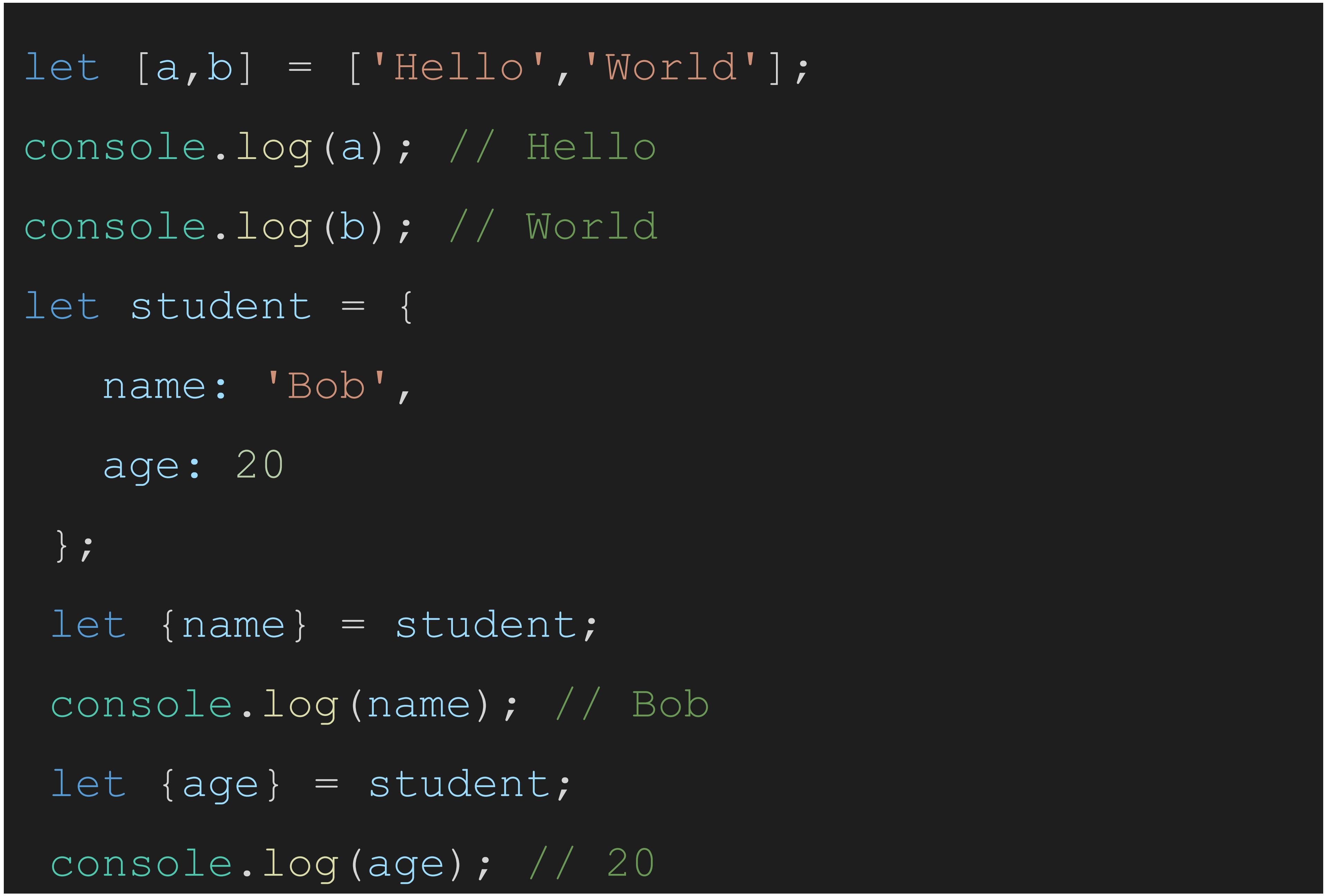
Used to split up array elements or object properties. So we can expand, copy an array, or clone an object.

```
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.



```
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
```

Shorthand syntax

if you want to define an object who's keys have the same name as the variables passed-in as properties, you can use the shorthand and simply pass the key name.

```
let cat = 'Miaow';
let dog = 'Woof';
let bird = 'Peet peet';

let someObject = {
  cat~~: cat~~,
  dog~~: dog~~,
  bird~~: bird~~
}

console.log(someObject);
```

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 => {return element > 10});

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