ECMAScript 2015+

2016, 2017, 2018 and more

Introduction

Ecma international - standards for technologies

ECMAScript (ECMA-262) is a scripting language specification created to standardize JavaScript.

- JavaScript
- ActionScript

ES5 and Vanilla Javascript

- The oldest and most compatible by all web browsers from Internet Explorer 6
- Specification implemented best by vanilla JavaScript
- Most commonly known as JavaScript
- Works natively on web browsers

ES2015 and beyond

- New specification introduced in 2015
    - ES2015, ES2016, ES2017, ES2018
- Introduces Classes, Promises, Block Scoping and more
- May not run natively in web browsers
    - Needs to be **transpiled** into ES5 to run
- Backwards compatible
- Node.js recommended

How-to Node.js

Steps to use Node.js

- Create a JavaScript file named `index.js`
- Add JavaScript code in the file
- In a terminal, run the following command:

```
$ node index.js
```

Variables and Constants

ES5:

```
var myVariable = 10;
```

ES2015:

```
let myVariable = 10;

const myVariable = 10;
```

Block Scoping and **Hoisting**

Loops

ES5:

```
for (var i = 0; i < 10; i++){
  console.log(i);
}
for (var i in array) { // i is the key of the array values
  console.log(i, array[i]);
}
```

ES2015:

```
for (let i of array){ // i is the value of the array
  console.log(i);
}
```

Functions

```
function myFunc(val = 'myDefaultValue') { // Default Values

  console.log(val);

}

myFunc();

myFunc('anotherValue');
```

Functions (Fat Arrow Representation)

Arrow Function or Lambda Expression

```
const myFunc = (val = 'myDefaultValue') => {
  console.log(val);
}
const getSquare = (num = 10) => num * num;
const log = str => console.log(str);
```

Object Destructuring

```
const myFunc = ({ key1 = 0, key2 = 0 }) => console.log(key1, key2);
const myObj = {
  key1: 10,
  key2: 20,
}
myFunc(myObj);
myFunc({ key1: 20, key3: 30, key4: 20 });
```

Array Destructuring

```
const myArray = [1, 2, 3, 4, 5];

const [a, b, c, d, e] = myArray;

console.log(a, b, c, d, e);
```

Spread Operation

```
const myArray = [1, 2, 3, 4, 5];
const [a, ...b] = myArray;
console.log(a, b);

function myFunc(val, ...args) {
  console.log(val, args);
}
myFunc(1, 2, 3, 4);
```

Array methods

- `includes: [1, 2, 3, 4, 5].includes(5);`
- `from: Array.from('Hello');`
- `findIndex: [1, 2, 3, 4, 5].findIndex(val => val === 3);`

Strings - Template Literals

ES5:

```
var myFunc = function(val) {
  return 'This is a ' + val;
}
```

ES2015:

```
const myString = `This is a string`;
const myFunc = val => `This is a ${val}`;
myFunc('string');
```

Classes

```
class MyClass{
  constructor(myVal) {
    this.myVal = myVal;
  }
  getVal() {
    return this.myVal;
  }
}
```

```
const myClass = new MyClass(10);

myClass.getVal();
```

Classes Inheritance

```
class MyClass{
  constructor(myVal) {
    this.myVal = myVal;
  }
  getVal() {
    return this.myVal;
  }
}
```

```
class AnotherClass extends MyClass {
  constructor(val) {
    super(100);
    this.val = val;
  }
  getNewVal() {
    console.log('My Variable', this.val);
    console.log('Inherited Class', this.myVal);
  }
}

const myClass = new AnotherClass(20);
myClass.getNewVal();
```

Promises

```javascript
const getVal = () => new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(100);
    // reject(0);
  }, 5000);
});

getVal()
  .then(value => console.log('Value', value))
  .catch(error => console.log('Error', error));
```

More Stuff

- Sets
- Async / Await Functions
- Object Spread
- Object.entries()
- Async Iteration
- Generators
- Transpiling (Babel)
- Bundling (Webpack)

That’s all folks