ECMAScript Features: ES6, ES7, ES8, ES9, ES10, ES11, ES12 – (Updated 2021)

BY QUIKIEAPPS OCTOBER 19, 2020REACTJS

ECMA stands for European Computer Manufacturer's Association. ECMAScript is a standard that defines the specification for scripting languages like JavaScript, ActionScript, JScript, etc. It is commonly used for the client-side scripting on the World Wide Web to affect how they look or behave for the user and due to its increasing popularity, it is increasingly being used for writing server applications and services using NodeJS and its frameworks like ExpressJS, etc.

JavaScript and EcmaScript

JavaScript is the programming language that is commonly used on the web to create and control dynamic website content. The core features of JavaScript are based on the ECMAScript standard and along with that JavaScript has some additional features that are not in the ECMAScript standard. Every browser has a built-in JavaScript interpreter where the JavaScript code runs. For example, Chrome has a built-in engine called the V8 engine where the JavaScript code is executed.

Comparison of ES5 and ES6:

ES5 or EcmaScript version 5 is the older version of the javascript which was introduced in 2009 whereas ES6 is the newer version which was introduced in 2015. There were some major changes in the ES6 version which introduced many new features like arrow function, classes Promise, etc. in JavaScript. Since 2015, each year newer versions are released with some additional features which we will discuss soon in this post. As of now, most of the browsers don't fully support the ES6 features. So transpilers like Babel are used which can convert the ES6 code to ES5.

Note: **ES6 = ECMAScript6 = ECMAScript 2015 = ES2015**

ES5 = ECMAScript 2009

ES6 (2015): ES6 was the first and significant update to the language since EcmaScript 2015 which was standardized in 2015.

ES6 has the following new features compared to ES5:

- **Arrows**: Arrows(=>) functions syntax is a shorthand for the ES 5 function syntax which are similar to the syntax of C#, Java 8, and CoffeeScript for the related feature. Unlike functions, arrows share the same lexical this as their surrounding code. Let's have a look at how we can use the arrow function:

Normal function e.g.

```
1 function planet() {
2 return "Earth";
3 }
```

Using Arrow function, this can be shortened to:

```
planet = () => "Earth";
```

It is not important to write return keywords and curly brackets if the arrow function has only a single statement.

Arrow functions do not have their own this because of which they are not well suited for defining object methods. Also, arrow functions are not hoisted, hence they must be initialized before they are declared.

- **Classes:** Classes in ES6 are similar to the prototype-based Object-Oriented pattern. Class patterns are easier to use because they have a single convenient declarative form as well as encourage interoperability. Classes not only support prototype-based inheritance but also static methods, super calls, constructors, and instances.
- **Promises:** A promise represents the processes that are already happening. A promise can be in one of the following steps: Pending, resolved, rejected. Initially, when the function execution starts, it is in a pending state. Once the code execution is completed, the promise is either resolved or rejected. Let's have a look at how promises can be used:

```
let promise = new Promise(function(resolve,reject)) {
resolve("Promise is resolved!!");
reject(new Error("Promise rejected due to error"));


promise.then(

      function(success);  // executed if promise is resolved

      function(failure);  // executed if promise is rejected
```

- **Module Loaders:** They support State isolation, Dynamic loading, Global namespace isolation, Compilation hooks as well as Nested virtualization.
- **Template Strings**: These allow us to construct strings syntactically. We can add a tag if we wish to make our string construction customized.

```
// Basic literal string creation
`In JavaScript '\n' is a line-feed.`

// Multiline strings
`In JavaScript, this is
not valid.`
```

```
// String interpolation
var name = "Bibek", day= "today";
`Hello ${name}, how are you ${day}?`
```

- **Destructuring:** It is a JavaScript expression that makes it possible to use values from arrays, or properties from objects, into distinct variables. We can easily use Rest or Spread operators to update an array of objects using the destructuring property.
- **Rest parameter(...):** The rest parameter (...) allows a function to treat an indefinite number of arguments as an array. This is mainly useful when we don't know the exact number of parameters that will be received.

```
function len(...args) {

    return len.length;
}

let x = length(4, 9, 16, 25, 29, 100, 66, 77);
```

- **let and const: let** is similar to var except that it is block-scoped whereas var has global scope. **Const** is used for a single assignment and it is also block scoped. **let** and **const** are not hoisted in the global scope.

```
function f() {
  {
    let x;
    {
      // okay, block scoped name
      const x = "sneaky";
      // error, const
      x = "foo";
    }
    // error, already declared in block
    let x = "inner";
  }
}
```

ECMAScript7 (ES7 – 2016):

ECMAScript 7 (2016) included only two new features which provided easy alternatives to the pre-existing functionalities. Since ES6 included all the many new features as it was a major update after

ES5(2009), there were only two updates in the ES7 version.

- **Exponentiation operator:** Along With supporting the basic arithmetic operations, ES7 finally introduced the exponentiation operator, “**”. It does the same job as that of the “MAth.pow()” function, i.e. returning the value of the first argument raised to the power of the second argument.
- **Array.prototype.includes():** This function checks for the value passed as an argument. If the array contains the value then this function will return true else it returns false.

ECMAScript 8 (ES8 – 2017) Features:

Some of the major features of EcmaScript 8 are listed below:

- **Async Functions:** Promises and Generators can be combined to perform asynchronous operations with the help of synchronous-looking code. Internally, async functions work much like generators. However, they are not translated to generator functions.

```
async function timeTakingFunction() {
  let functionResponse= new Promise(function(myResolve, myReject) {
setTimeout(function() { myResolve("I will be executed after 3 seconds."); },
3000);
  });
  console.log(functionResponse);
}

timeTakingFunction();
```

ECMAScript 9 (2018) Features:

Some of the major features of EcmaScript 9 are listed below:

- **Asynchronous Iteration:** EcmaScript 2018 introduced asynchronous iterators and iterables. This will help in reading text from an HTTP connection very effortlessly. A new IterationStatement known as for–await–of was introduced that also adds syntax that can help in creating async generator functions.
- **Promise.prototype.finally:** As we know, promises make the execution of the callback function much easier. Numerous promise libraries have a method called “finally” by which we can run the code and it won’t matter how the Promise gives a resolution. When a promise gets fulfilled or denied, it registers a callback that gets invoked and that callback can be called in the finally block. E.g.

```
let myPromise = new Promise();
myPromise.then();
myPromise.catch();
myPromise.finally();
```

- **Object Rest Properties:** This is one of the important feature of Ecmascript 2018. This allows us to destruct an object and collect the remaining elements of the object into a new object. For e.g.

```
let { a, b, ...c } = { a: 1, b: 2, c: “Hello World”, d: 4 };

console.log(a); // 1

console.log(b); // 2

console.log(c); // { c: “Hello World”, d: 4 }
```

ECMAScript 10 (2019) Features:

Its important features are listed below:

- **Array.flat():** This recursively flattens your array up to a level that is specified to it. This is a great addition because it helps users by avoiding the use of external libraries to write their own code for such a simple task. It can flatten the array any number of times that is mentioned to it.
  - **Array.flatmap():** As the name “flatmap” suggests, it allows us to map through an array containing items and flatten the array in one go. For the ‘map’ function we will get a 2 dimensional result for an array but when we use the Array.flatmap() function, we will obtain a flattened one-dimensional array as a result of using it.
- **String.trimStart/trimEnd:** This is an update of the string prototype. It helps us trim the white spaces from the beginning or end using trimstart or trimend.

ECMAScript 11 (2020) Features:

Its important features are listed below:

- **import():** This version provides us with a dynamic import() which gives promise in return for the module namespace object of the module that was requested. Thus, now it's possible to assign imports to a variable easily by use of async or await.
- **globalThis:** For solving issues of working with multiple platforms, globalThis is a unique solution provided in this new version. This avoids platform-specific globals and works on all platforms.
- **Nullish Coalescing Operator (??):** This operator checks whether the property presented is null or undefined and that is not for all false values. If it is not a null or undefined value, the operator returns the initial value or prints the property value.

- **Optional Chaining Operator (?.):** It checks whether the user property is available or not. The chaining operator provides a much cleaner way of doing that.

ECMAScript 12 (ES12 – 2021) Features:

Its important features are listed below:

- **replaceAll:** This method is introduced to replace all occurrences of a string with another string value. Initially, there was replace method which replaces only the first occurrence of a string with some other value while replaceAll can be used to replace all the instances. Let's take an example to understand this.

```
let string = "Javascript is used in front-end development. Javascript is also used in backend development using NodeJS"

// replace method
console.log(string.replace("Javascript", "Typescript"));
// Output: Typescript is used in front-end development. Javascript is also used in backend development using NodeJS

// new replaceAll method
console.log(string.replaceAll("Javascript", "Typescript"));
// Output: Typescript is used in front-end development. Typescript is also used in backend development using NodeJS
```

- **Numeric separator:** Dealing with large numbers can create some confusion while reading the code. For e.g. if I write "9000000000", probably no one will read/know the value of this number. So, this version of EcmaScript introduced a numeric separator. The above number can be written as "9_000_000_000". This is the same as the previous value and we can simply say read its value is 9 billion.

We can also check if that is valid syntax in the following way:

```
const number = 9000000000;
console.log(number === 10 ** 9);
// Output: true
```

- **Private Methods:** This feature helps to create methods in a class that can only be accessible to the methods where they are defined. Let's take an example to understand this in a much better way.

```
class PrivateMethods {

// Private method
#privateName() {
console.log("I am available only in methods of this class");
}

// Public method
publicName() {
this.#privateName();
console.log("I am available to all objects of this class");
}
}

const method = new PrivateMethods();
method.publicName();
// Output: I am available only in methods of this class
// I am available to all objects of this class

method.privateName();
// Error: method.privateName is not a function.
```

- **Promise.any() :** Promise.any is almost opposite to Promise.all. Promise.all is executed when all the promises that are mentioned while defining it are resolved whereas Promise.any is executed if any of the promises that is mentioned is resolved. Let's take an example to understand this.

```
const first= new Promise((resolve, reject) => {
  reject("I am rejected");
});

const second= new Promise((resolve, reject) => {
```

```
  setTimeout(resolve, 500, "I will resolve in 500ms");
});

const third= new Promise((resolve, reject) => {
  setTimeout(resolve, 100, "I will resolve in 100ms");
});

Promise.any([first, second, third]).then((response) => {
  console.log(response);})

// Output: I will resolve in 100ms
```

In the above example, we can see that the promise third will be executed first as it will be resolved after 100ms while the promise second will be resolved after 500ms whereas the promise first always gets rejected. But what if all the promise gets rejected. Well, in this case, we can use a try...catch block to catch if all the promises are rejected or use then and catch chain to catch the errors. Let's look at another example to see how we can deal with such scenarios.

```
const first= new Promise((resolve, reject) => {
  reject("I am rejected");
});

const second= new Promise((resolve, reject) => {
  setTimeout(reject, 500, "I will be rejected in 500ms");
});

const third= new Promise((resolve, reject) => {
  setTimeout(resolve, 100, "I will be rejected in 100ms");
});

Promise.any([first, second, third]).then((response) => {
  console.log(response);}).catch(err => console.log(err))

// Output: Aggregate Error: No Promise in Promise.any was resolved
```

Conclusion:

Each year Ecmascript releases some features that help to improve the quality and we must be familiar with the latest features to be able to use all the latest features. From its establishment in 1995 to the present day, there have been major changes that have helped ECMAScript gain its present popularity among developers. JavaScript and JScript are both made to be compatible with ECMA and it's still been updated through numerous proposals of development.