

JavaScript New Features

Mendel Rosenblum

ECMAScript

- New standard for ECMAScript released yearly
 - Relatively easy to get a new feature into the language
- Transpiling: Translate new language to old style JavaScript
 - Allows front-end software to be coded with new features but run everywhere.
 - For example: [Babel](#). Check out: <https://babeljs.io/en/repl> new JS in -> old JS out
- Frontend frameworks are aggressively using new language features
 - React.js - Encourages use of newer ECMAScript features
 - Angular - Encourages Typescript - Extended JavaScript with static types and type checking

Lots of new features in ECMAScript

- Already seen a few
 - `let`, `const`, `class`, `=>`
- Here are a few more you might encounter:
 - Modules
 - Default parameters
 - Rest parameters ...
 - Spread operator ...
 - Destructuring assignment
 - Template string literals
 - `Set`, `Map`, `WeakSet`, `WeakMap` objects, async programming

Modules - Variables global to a file not system

Old Way

```
var exportedName =  
  (function () {  
    var x, y, x;  
    ...  
    return {x: x, y: y};  
  })();
```

Use Immediately Invoked Function Expressions using closures to make module variables function scope and only return a single object to access them.

New Way

```
var x, y, x;  
...  
var exportedName = {x: x, y: y};  
  
export exportedName;
```

Can explicitly define file's exports and then import the module in another file. Two common ways:

- Common.js (Node.js):
module.exports/require()
- ECMAScript 6: export/import

Default parameters - Parameters not specified

Old Way

```
function myFunc(a,b) {  
    a = a || 1;  
    b = b || "Hello";  
}
```

Unspecified parameters are set to undefined.
You need to explicitly set them if you want a different default.

New Way

```
function myFunc (a = 1, b = "Hello") {  
  
}
```

Can explicitly define default values if parameter is not defined.

Rest parameters . . .

Old Way

```
function myFunc() {  
    var a = arguments[0];  
    var b = arguments[1];  
    var c = arguments[2];  
    arguments[N]  
    //  
}
```

Parameters not listed but passed can be accessed using the arguments array.

New Way

```
function myFunc (a,b,...theArgsArray) {  
    var c = theArgsArray[0];  
}
```

Additional parameters can be placed into a named array.

Spread operator . . .

Old Way

```
var anArray = [1,2,3];  
myFunc.apply(null, anArray);
```

```
var o = [5].concat(anArray).concat([6]);
```

Expand an array to pass its values to a function or insert it into an array.

New Way

```
var anArray = [1,2,3];  
myFunc(...anArray);
```

```
var o = [5, ...anArray, 6];
```

Works on iterable types: strings & arrays

Destructuring assignment

Old Way

```
var a = arr[0];  
var b = arr[1];  
var c = arr[2];
```

```
var name = obj.name;  
var age = obj.age;  
var salary = obj.salary;
```

```
function render(props) {  
  var name = props.name;  
  var age = props.age;
```

New Way

```
let [a,b,c] = arr;
```

```
let {name, age, salary} = obj;
```

```
function render({name, age}) {
```


Template string literals

Old Way

```
function formatGreetings(name, age) {  
  
  var str =  
    "Hi " + name +  
      " your age is " + age;  
  ...  
}
```

Use string concatenation to build up string from variables.

New Way

```
function formatGreetings(name, age) {  
  
  let str =  
    `Hi ${name} your age is ${age}`;  
}
```

Also allows multi-line strings:

```
`This string has  
two lines`
```

Very useful in frontend code. Strings can be delimited by `"`, `'`, or ```

For of

Old Way

```
var a = [5,6,7];  
var sum = 0;  
for (var i = 0; i < a.length; i++) {  
    sum += a[i];  
}
```

Iterator over an array

New Way

```
let sum = 0;  
for (ent of a) {  
    sum += ent;  
}
```

Iterate over arrays, strings, Map, Set, without using indexes.

EcmaScript 2020

- Nullish coalescing operator (??)
 - `val1 ?? val2;` - Returns `val2` if it is null or undefined otherwise `val1`, like `val1 || val2`
 - `param ?? 32;` - Works for all number values of `param` including 0
- Optional Chaining (?.)
 - `obj?.prop` - Returns undefined if `obj` is undefined or null otherwise returns `obj.prop`
 - `obj?.subobj?.prop;` - Handles `obj` or `subobj` being null or undefined.
 - `func?.();` - Calls `func` only if not null or undefined.
 - `arr?.[1]` - Access array only if `arr` is not undefined or null.
- BigInt - Bigger than 53bit integers - Example: `9007199254740992n`
 - Makes working with 64bit integers from other languages easier

Some additional extensions

- Set, Map, WeakSet, WeakMap objects
 - Defined interfaces for common abstractions
- `async/await` and Promises
 - Asynchronous programming help