ES2015+ cheatsheet

A quick overview of new JavaScript features in ES2015, ES2016, ES2017 and beyond.

Block scoping

Let

```
function fn () { let x = 0 if (true) { let x = 1 // only inside this `if` } }
```

Const

```
const a = 1
```

`let` is the new var. Constants work just like `let`, but can't be reassigned. See: Let and const

Backtick strings

Interpolation

```
const message = `Hello
```

Multiline strings

```
const str = ` hello wor
```

Templates and multiline st

New methods

New string methods

```
"hello".repeat(3) "hello".includes("ll")
"\u1E9B\u0323".normalize("NFC")
```

See: New methods

Binary and octal literal

```
let bin = 0b1010010 let
```

See: Binary and octal litera

Exponent operator

```
const byte = 2 ** 8 // Same as: Math.pow(2, 8)
```

Classes

```
class Circle extends Sh
```

Constructor

```
constructor (radius) {
```

Methods

```
getArea () { return Mat
```

Calling superclass methods

```
expand (n) { return sup
```

Static methods

```
static createFromDiamet
Circle(diameter / 2) }
```

Syntactic sugar for prototy

Promises

Making promises

```
new Promise((resolve, reject) => { if (ok) {
resolve(result) } else { reject(error) } })
```

For asynchronous programming. See: Promises

Using promises

```
promise .then((result) => { ··· }) .catch((error)
=> { ··· })
```

Async-await

```
async function run () { const user = await
getUser() const tweets = await getTweets(user)
return [user, tweets] }
```

async functions are another way of using functions.

See: async function

Destructuring

Destructuring assignment

Default values

Arrays

```
const [first, last] = ['Nikola', 'Tesla']
```

Objects

```
let {title, author} = { title: 'The Silkworm',
author: 'R. Galbraith' }
```

Supports for matching arrays and objects. See: Destructuring

Loops

```
for (let {title, artist} of songs) { ··· }
```

The assignment expressions work in loops, too.

```
const scores = [22, 33] const [math = 50, sci = 50,
arts = 50] = scores

// Result: // math === 22, sci === 33, arts === 50
```

Default values can be assigned while destructuring arrays or objects.

Reassigning keys

```
function printCoordinates({ left: x, top: y }) {
console.log(`x: ${x}, y: ${y}`) }

printCoordinates({ left: 25, top: 90 })
```

This example assigns x to the value of the `left` key.

Spread

Object spread

with Object spread

```
const options = { ...defaults, visible: true }
```

without Object spread

```
const options = Object.assign( {}, defaults, {
visible: true })
```

The Object spread operator lets you build new objects from other objects.

See: Object spread

Array spread

with Array spread

```
const users = [ ...admi
```

without Array spread

```
const users = admins .c
'rstacruz' ])
```

The spread operator lets y
way.

See: Spread operator

Functions

Function arguments

Default arguments

```
function greet (name = 'Jerry') { return `Hello ${name}` }
```

Rest arguments

```
function fn(x, ...y) { // y is an Array return x * y.length }
```

Spread

```
fn(...[1, 2, 3]) // same as fn(1, 2, 3)
```

Default, rest, spread. See: Function arguments

Fat arrows

Fat arrows

```
setTimeout(() => { ···
```

With arguments

```
readFile('text.txt', (e
```

Implicit return

```
numbers.map(n => n * 2)
implicit return // Same
(n) { return n * 2 })
```

Like functions but with thi

Objects

Shorthand syntax

```
module.exports = { hello, bye } // Same as: module.exports = { hello: hello, bye: bye }
```

See: Object literal enhancements

Methods

```
const App = { start ()
// Same as: App = { sta
```

See: Object literal enhance

Getters and setters

```
const App = { get closed () { return this.status === 'closed' }, set closed (value) { this.status = value ? 'closed' : 'open' } }
```

See: Object literal enhancements

Computed property n

```
let event = 'click' let
[`on${event}`]: true }
'onclick': true }
```

See: Object literal enhance

Modules

Imports

```
import 'helpers' // aka: require('···')
```

```
import Express from 'express' // aka: const Express
= require('···').default || require('···')
```

```
import { indent } from 'helpers' // aka: const
indent = require('···').indent
```

```
import * as Helpers from 'helpers' // aka: const
Helpers = require('···')
```

```
import { indentSpaces as indent } from 'helpers' //
aka: const indent = require('···').indentSpaces
```

`import` is the new `require()`. See: Module imports

Exports

```
export default function
module.exports.default
```

```
export function mymetho
module.exports.mymethod
```

```
export const pi = 3.141
= ...
```

`export` is the new `module.`

Generators

Generators

```
function* idMaker () { let id = 0 while (true) {
yield id++ } }
```

```
let gen = idMaker() gen.next().value // → 0
gen.next().value // → 1 gen.next().value // → 2
```

It's complicated. See: Generators

For..of iteration

```
for (let i of iterable)
```

For iterating through gene
iteration