ES2015+ cheatsheet

Proudly sponsored by
Gitbook: *Write your JavaScript skills to grow*
Open Source, Open Knowledge
Open Source, Open Collaboration

A quick overview of new JavaScript features (ES2015, ES2016, ES2017, ES2018 and beyond).

Block scoping

```
let name = "Jeremy";
name = "Jeremy";
```

```
const PI = 3.14;
PI = 3;
```

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

Backticks strings

```
const message = `Hello I name`;
```

Interpolation

```
const name = "Jeremy";
const greeting = `Hello, my name is ${name}!`;
```

Multiline strings

```
console.log(`string text line 1
string text line 2`);
```

Tagged templates and multiline strings. See: Template strings.

Binary and octal literals

```
let bin = 0b1010010;
let oct = 0o755;
```

See: Binary and octal literals.

New methods

New string formats

```
Number("0111.1");
Number("0b111.1");
Number("0o111.1");
Number("foo");
Number.EPSILON;
Number.isFinite(Infinity);
Number.isNaN("foo");
Number.parseFloat("0.111");
Number.parseInt("011", 10);
```

See: New methods

Classes

```
class Circle extends Shape {
  constructor(id, x, y, radius) {
    super(id, x, y);
    this.radius = radius;
  }

  toString() {
    return `Circle(${this.id}, ${this.x}, ${this.y}, ${this.radius})`;
  }

  static fromObject(id, x, y, obj) {
    return new Circle(id, x, y, obj.radius);
  }
}
```

Classes

See: Classes

Exponent operator

```
let byte = 2 ** 8;
let megabyte = byte ** 2;
let gigabyte = megabyte ** 2;
let terabyte = gigabyte ** 2;
```

Exponent operator

Promises

Making promises

```
function fetchURL(url, timeout, onresolve, onreject) {
  let timeoutFn = null, request = null
  let clearAll = function() {
    clearTimeout(timeoutFn);
    if (request && request.hasOwnProperty('abort'))
    request.abort();
  }

  timeoutFn = setTimeout(function() {
    clearAll();
    if (onreject)
    onreject(new Error("Timeout fetching " + url));
  }, timeout);

  if (typeof window.fetch == "function")
  {
    var isClosed = false;
    fetch(url).then(function(response) {
      if (!isClosed)
      {
        clearTimeout(timeoutFn);
        if (onresolve)
        onresolve(response);
      }
    }, function(err) {
      if (!isClosed)
      {
        clearTimeout(timeoutFn);
        if (onreject)
        onreject(err);
      }
    });
    return function() {
      isClosed = true;
      clearAll();
    }
  }

  var req = new XMLHttpRequest();
  req.onreadystatechange = function() {
    if (req.readyState == 4)
    {
      clearTimeout(timeoutFn);
      if (req.status == 200)
      {
        if (onresolve)
        onresolve(req);
      }
      else
      {
        if (onreject)
        onreject(new Error("Error fetching " + url));
      }
    }
  };
  try
  {
    req.open("GET", url);
    req.timeout = 0;
    req.send();
  }
  catch(e)
  {
    if (onreject)
    onreject(e);
  }
  return function() {
    req.abort();
    clearAll();
  }
}
```

For asynchronous programming. See: Promises.

Using promises

```
fetchURL("/data.txt", 5000, function(resp) {
  console.log("Fetched data successfully");
}, function(err) {
  console.log("Error fetching data: " + err);
});
```

Using promises with finally

```
fetchURL("/data.txt", 5000, function(resp) {
  console.log("Fetched data successfully");
}, function(err) {
  console.log("Error fetching data: " + err);
}).finally(function() {
  console.log("Finished");
});
```

The handler is called when the promise is fulfilled or rejected.

Promise functions

```
Promise.all([p1, p2, p3]).then(function(values) {
  console.log("Resolved", values);
});
Promise.race([p1, p2, p3]).then(function(value) {
  console.log("Resolved", value);
});
var p4 = new Promise(p1);
var p5 = Promise.resolve("Hello");
var p6 = Promise.reject(new Error("Bad"));
var p7 = Promise.resolve(3);
var p8 = Promise.reject(new Error("Bad"));
```

Promise.all([p1, p2, p3]).then(function(values) {
console.log("Resolved", values);
});
Promise.race([p1, p2, p3]).then(function(value) {
console.log("Resolved", value);
});
var p4 = new Promise(p1);
var p5 = Promise.resolve("Hello");
var p6 = Promise.reject(new Error("Bad"));
var p7 = Promise.resolve(3);
var p8 = Promise.reject(new Error("Bad"));