# ECMAScript 2015

- `let`, `const` (define block-local vars), `function` in block scope
- destructuring: `let {x, y} = pt; let [s, v, o] = triple();` (assuming `let pt = {x:2, y:-5}`, for instance).
- parameter default values: `function f(x, y=1, z=0) {…}`
- rest: `function g(i, j, ...r) { return r.slice(i, j); }` (instead of using `arguments` like a madman).
- spread: `let a = [0,1,2,3], o = new Something(...a);`. Also works in array literals: `[1, ...array, 4]`.
- Shorthand object literals: `let one = 1; { one, func_one() {return 1;}, ['key ' + one]: 1 }`.
- fat arrow: `(a) => a * a` is the same as `(function(a) { return a * a; }).bind(this)`
- map, set: `let m = new Map(); m.set(key, value); m.has(key); m.get(key)`. Also has `.clear()`, `.delete()`, `.forEach()`, `.keys()`.
- weak map: `let map = new WeakMap()`. Use it if you have circular references in it. You also have `new WeakSet()`.
- promise: `new Promise((resolve, reject) => {…})`.
  - `resolve(valueOrPromise)` returns the promised value (or a new promise, to chain them) in `promise.then(value => {…})`.

  - `reject(new Error(…))` breaks the promise in `promise.then(…).then(…).catch(error => {…})`.
  - Quick promise creation: `Promise.resolve(value)`, `Promise.reject(error)`.
  - Iterables: `Promise.all(listOfPromises).then(listOfValues => …)`, `Promise.race(listOfPromises).then(valueThatResolvedFirst => …)`

- proxies: `let obj = new Proxy(proto, handler)`. Long story short: ~ operator overloading with object-like elements. (Can also catch all property accesses on an object.)
- generators: `function* gen() { yield 1; yield 2; }` Actually, `gen()` returns ~ an object with a `next()` function property.
- iterators: `for (var [key, val] of items(x)) { alert(key + ',' + val); }`. Iterators can be generators or proxies.
- class syntax, with `extends`, `super`, and `static`:

```
class Point extends Base {
  constructor(x,y) {
    super();
    this[px] = x, this[py] = y;
    this.r = function() { return Math.sqrt(x*x + y*y); }
  }
  get x() { return this[px]; }
  get y() { return this[py]; }
  proto_r() { return Math.sqrt(this[px] * this[px] +
      this[py] * this[py]); }
  equals(p) { return this[px] === p[px] &&
```

```
        this[py] === p[py]; }
}
```

- Symbols generate unique inaccessible keys, useful in maps and classes (private members).

```
let a = Map();
{
  let k = Symbol();
  a.set(k, 'value');
  // Here, we can get and reset 'value' as a.get(k).
}
// Here, a.get(k) is invalid, a.size is 1, but the key cannot be seen.
```

- modules:

```
module math {
  export function sum(x, y) {
    return x + y;
  }
  export var pi = 3.141593;
}

import {sum, pi} from math;
alert(sum(pi,pi));
```

- template strings: multiline, substitution-ready strings with extensibility. `` `You are ${age} years old.` ``.

```
// The following regexp spans multiple lines.
re`line1: (words )*
line2: \w+`

// It desugars to:
re({raw:'line1: (words )*\nline2: \w+',
    cooked:'line1: (words )*\nline2: \w+'})
```

- Typed Array

---

Specification: http://people.mozilla.org/~jorendorff/es6-draft.html

Proposals: http://wiki.ecmascript.org/doku.php?id=harmony:proposals

Thanks Brendan Eich, Allen Wirfs-Brock, Douglas Crockford, Mark Miller, Tom Van Cutsem, Alex Russell, and so many others.