ECMAScript es un lenguaje vivo. Cada año se suceden nuevas versiones con multitud de nuevas funcionalidades. Actualmente, los encargados de decidir qué nuevos cambios se introducen es el TC39. Ingenieros de Google, Airbnb, Twitter, Facebook y muchas otras compañía, componen este comité.

Nuevas versiones

Hasta ahora, ECMAScript publicaba sus nuevas versiones siguiendo una secuencia numérica: *ES5*, *ES6*… A partir de ahora, cada año se liberará una versión incluyendo el propio año como indicador.

Proceso de aceptación de funcionalidades

Podríamos formular este enunciado como una pregunta, ¿cómo se convierte una propuesta en funcionalidad oficial? Si os fijáis en la última columna del repositorio de propuestas (*TC39 Proposals*), esta indica la fase actual de la funcionalidad. **Para que una propuesta llegue a ser implementada en e lenguaje, debe de pasar por las 5 fases** definidas por el TC39:

- 1. Strawman
- 1. Proposal
- 1. Draft
- 1. Candidate
- 1. Finished

Si os interesa este tema, podéis leer el documento de procesos del TC39.

Compatibilidad

ECMAScript avanza cada año, pero no todo depende del TC39. Los navegadores tienen la última palabra sobre que funcionalidades se incluyen y cuando lo harán.

Además, las personas que utilizarán nuestro aplicación pueden decidir no actualizar sus navegadores. **Esto es algo que se escapa inevitablemente de nuestro control**.

Sin embargo, uno de los **requisitos para del TC39 es que las propuestas sean retrocompatibles**. Gracias a esto podemos utilizar funcionalidades de las nuevas versiones sin romper la compatibilidad con los navegadores más antiguos.

Polyfills y BabelJS

> A shim that mimics a future API providing fallback functionality to older browsers.
> - @paul_irish

Un *polyfill* es una pequeña **librería que simula nuevas funcionalidades en navegadores antiguos**. Cuando la funcionalidad ya esta disponible de manera nativa, esta librería deja de aplicar sus cambios.

Incluyendo distintos *polyfills* simulamos nuevos métodos, pero esto no es suficiente. Es en este punto donde entra en juego BabelJS. Babel es un compilador de ECMAScript. **Este transforma nuestro código aumentando la compatibilidad**. Por ejemplo, el siguiente código toma cada elemento del array y le suma uno:

```
[1, 2, 3].map(n => n + 1);
```

No os preocupéis si no entendéis la sintaxis o si creéis que esto no es ~~JavaScript~~ ECMAScript, es normal. El resultado de la compilación de Babel es el siguiente:

```
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

Babel se ha convertido en una herramienta necesaria para nuestros proyectos, puesto que la gran mayoría de nuevas librerías hacen uso de los nuevos operadores, sintaxis y métodos. Entraremos más en detalle en los siguientes capítulos.