# UD1. INTRODUCCIÓN AL DESARROLLO WEB DEL LADO CLIENTE



# CREACIÓN DE APLICACIONES WEB

## 1.1. ¿QUÉ ES UNA APLICACIÓN WEB?

Una aplicación es un software que permite realizar una determinada tarea o servicio. Las aplicaciones de escritorio son las aplicaciones clásicas.

Las aplicaciones web, son aquellas que se han creado para ser ejecutadas por un software especial conocido como **navegador**. Con lo cual, los lenguajes en los que se pueden programar estas aplicaciones son aquellos que un navegador es capaz de traducir.

Hoy en día, las aplicaciones web son el tipo de aplicación más popular, ya que, independientemente del sistema, todos los usuarios tienen instalado en su equipo algún navegador (Chrome, IE, Firefox..).

## 1.2. PROCESO DE DESARROLLO DE APLICACIONES WEB

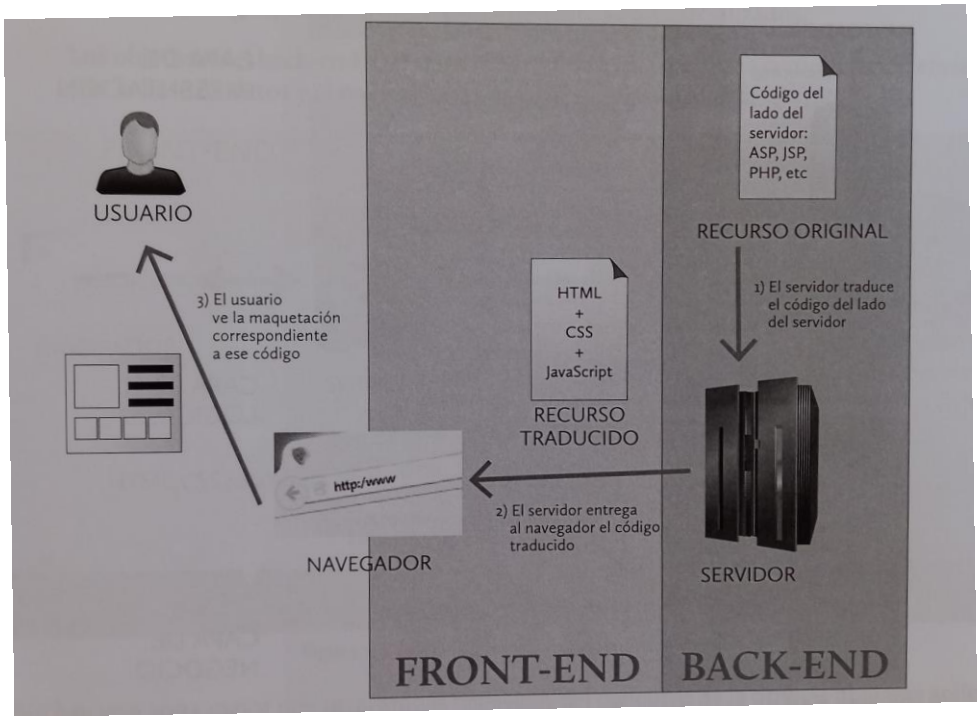
Al crear aplicaciones web podemos utilizar diferentes lenguajes y tecnologías, pero hay 2 tipos de tecnologías principales a tener en cuenta:

- ✓ **Lenguajes y tecnologías que se ejecutan en el lado cliente**. Se trata de los elementos que se incorporan junto al código HTML de una aplicación web y que necesitan ser interpretados por el navegador del usuario. Este código puede ser, incluso, examinado por el usuario.
- ✓ **Lenguajes y tecnologías del lado del servidor**. Es la programación de la aplicación escrita para ser interpretada por el servidor web. El código escrito, en este caso, queda oculto al usuario ya que se ejecuta de forma externa.

Una aplicación web utiliza ambas tecnologías.

Las aplicaciones web actuales utilizan lo que se conoce como arquitectura de 3 niveles (**three-tier architecture**), a veces incluso se habla de más capas. Estas son:

- **Capa de presentación, capa del cliente o capa del navegador**. Controla la parte de la aplicación web que llega al navegador del usuario. Es decir, se encarga de la forma de presentar la información al usuario. Esta capa es la relacionada con la programación del lado cliente o **front-end** y, actualmente se programa en HTML, CSS y JavaScript. El código que procede de esta capa lo envía el servidor web y lo entrega al navegador, esa comunicación se realiza por medio del protocolo http.
- **Capa lógica, capa del servidor o capa de aplicación**. Es la encargada de gestionar el funcionamiento interno de la aplicación. La capa se programa en lenguajes del lado del servidor como: PHP, ASP, Python, Java, JavaScript (hay JavaScript del lado servidor) y otros. El código de esta capa se traduce por un **servidor de aplicaciones** que, normalmente, es un servidor web con módulos de traducción de esos lenguajes.

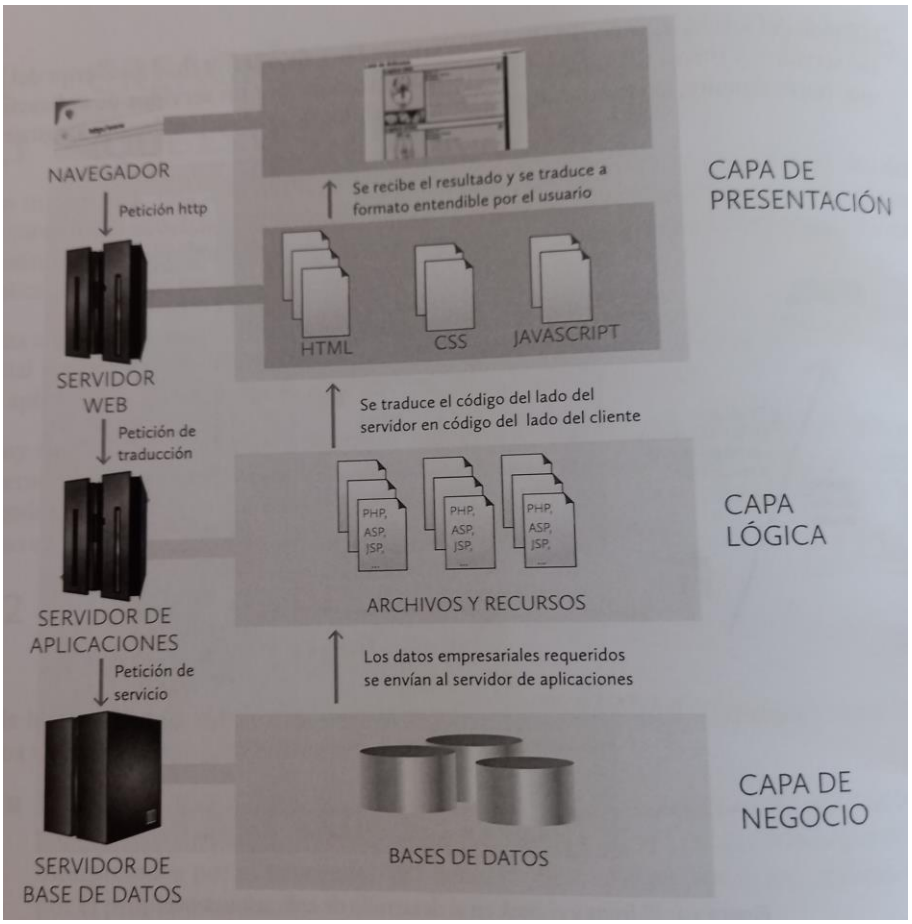



El front y el back en el desarrollo de aplicaciones web

El servidor de aplicaciones traduce ese código cuyo resultado será código compatible con la capa de presentación. Por ejemplo, si hemos escrito esta capa usando el lenguaje PHP, el servidor de aplicaciones traducirá el código PHP produciendo código HTML, CSS y JavaScript que es el que entregará al navegador. Al navegador no le llega el código PHP original.

- **Capa de negocio o capa de datos**. Se almacena la información empresarial. Esta información ha de quedar oculta a cualquier persona sin autorización. En esta capa, habitualmente, se encuentra un sistema **gestor de bases de datos (SGBD)**, que es un software que sirve para administrar la información de la empresa. También, puede haber otro tipo de servidores que sirvan para proporcionar recursos empresariales tales como vídeo, audio, certificados, correo, etc.

  La capa lógica se comunica con esta capa para obtener datos que luego procesa y entrega, ya convertido en un formato interpretable, al navegador. De modo que el proceso de acceso a estos recursos queda oculto totalmente al navegador, lo que añade mayor seguridad al proceso.




Funcionamiento de las 3 capas

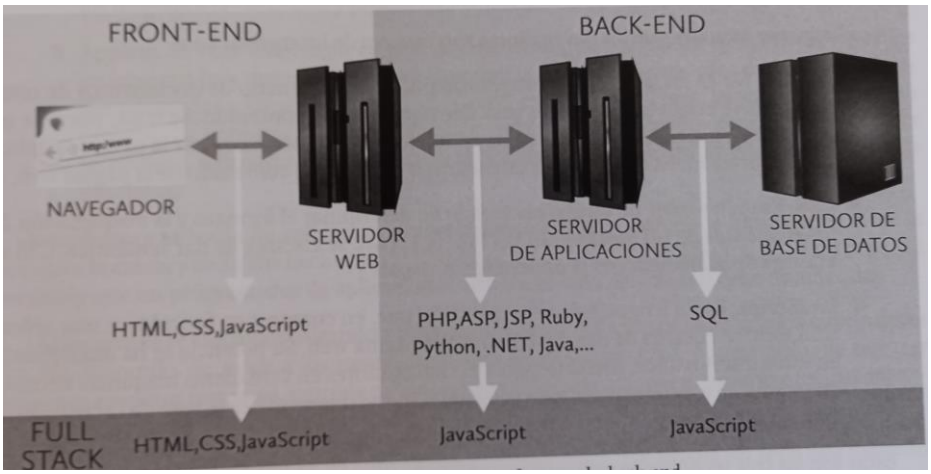
## 1.3. FRONT-END Y BACK-END

La división en capas del funcionamiento de las aplicaciones web ha propiciado 2 tipos fundamentales de desarrolladores de aplicaciones: los programadores **front-end** y los programadores **back-end**:

**Front-end:** Desarrollo encargado de producir la apariencia final de la aplicación que verá el usuario. Está relacionada con la capa de presentación del modelo de 3 capas. En el front-end trabajan los diseñadores de la aplicación, encargados de determinar lo que se conoce como experiencia de usuario (conocida por las siglas **UX**) y el diseño visual de la aplicación y los programadores de los lenguajes HTML, CSS y JavaScript que realmente construyen el funcionamiento final de la aplicación. El objetivo es optimizar la presentación y conseguir una interfaz final lo más agradable y funcional posible para los usuarios.

**Back-end:** Un desarrollador back-end se encarga de realizar la parte de la aplicación que queda oculta al usuario. El funcionamiento de la capa lógica y de negocio sería el back-end, por lo que en esta zona se usan los lenguajes del lado del servidor y los de base de datos. Esos lenguajes son muy variados y mucho más especializados.

Los objetivos del back-end son: un acceso rápido a los datos, una comunicación eficiente con la base de datos y el navegador, control de seguridad, etc.




Funcionamiento front-end y back-end

No hay una zona mejor que otra, ambas son necesarias. Al final una aplicación típica puede utilizar numerosos lenguajes y tecnologías y eso impide que un programador front-end tenga difícil pasar al back-end y viceversa. Para mitigar este problema, hay desarrollos que utilizan el mismo lenguaje en las 3 capas (por ejemplo JavaScript) y de este modo el cambio es más intuitivo. Se conoce a esta estrategia como **Full Stack Development**.

Parecía que esta estrategia iba a tener un gran resultado, y en parte lo ha tenido, pero lo cierto es que la especialización de profesionales en la creación de aplicaciones web sigue siendo muy importante e incluso cada vez se especializa más. Así, sigue siendo normal, a la hora de optar a un puesto profesional, que la oferta se refiera a un programador front-end o a un programador back-end. Es, todavía, extraño pedir un programador full-stack.

Incluso, muchas veces, se requieren profesionales más específicos como diseñadores administradores de bases de datos, especialistas en **DevOps**[1], desarrolladores de juegos, científicos de datos o especialistas en IA.

# TECNOLOGÍAS DEL LADO DEL CLIENTE

## 2.1. RECORRIDO POR LAS TECNOLOGÍAS FRONT-END

Los lenguajes actuales que los navegadores son capaces de interpretar son:

- **HTML.** No es un lenguaje de programación, sino un formato de documentos de texto que incluye etiquetas especiales para dar significado al contenido. Se trata, pues, de un lenguaje de marcado. Es la base de las aplicaciones Web y se encarga de la parte semántica de la aplicación. Es decir, de dar estructura conceptual al contenido de la página web.

[1] Práctica de los ingenieros informáticos que aglutina conocimientos informáticos tanto de administración de sistemas cmo de desarrollo web. Permite integrar todos los procesos, e implica: crear, probar e implementar el software en un entorno de producción.

- **CSS.** Es un lenguaje de diseño encargado de determinar el formato y la maquetación de los elementos de un documento HTML. Si HTML se encarga de dar semántica, CSS se encarga de la apariencia y la presentación visual.
- **JavaScript**. Es un lenguaje de programación que, en cuanto a su función en una aplicación web, se encarga de de dar dinamismo a la página web. Su potencia se ha multiplicado en estos últimos años, convirtiendo a los navegadores en verdaderas máquinas virtuales que traducen este lenguaje a toda velocidad. Indudablemente es la base de la creación de aplicaciones web.

  Estas son las 3 tecnologías principales y obligadas en toda creación de aplicaciones web en la actualidad. A lo largo del tiempo, han existido y existen otras tecnologías.

- **XML**. Hace unos años se consideró a XML como el lenguaje de marcado que acabaría con el propio HTML. No fue así, pero sigue siendo una tecnología a tener en cuenta. Se trata de un lenguaje de marcado que permite definir formatos documentales a voluntad y que se nutre de una serie de tecnologías anexas capaces de dar formato, dinamismo y otras capacidades a documentos XML. En la actualidad, en cuanto a la creación de aplicaciones, se utiliza principalmente como formato de intercambio de datos entre aplicaciones y servicios. La mayoría de navegadores comprenden este lenguaje.
- **JSON**. Es un formato documental procedente del lenguaje JavaScript y que, en gran medida, está reemplazando a XML como formato de intercambio de datos. Los navegadores no comprenden este formato de forma nativa, pero existen numerosos **plugins** (software que se añade al navegador para otorgarle funcionalidades extra) que permiten visualizar adecuadamente este tipo de documentos en el navegador.
- **SVG**. Es un lenguaje basado en XML que permite mostrar imágenes vectoriales en los navegadores. Actualmente está implantado y reconocido en todos los navegadores actuales.
- **Flash**. Es una tecnología que se creó para conseguir crear, de forma eficiente y rápida, aplicaciones web enriquecidas. Tuvo un enorme éxito desde finales de los años 90 del siglo pasado, pero actualmente está en claro desuso. Requería la instalación de un plugin en el navegador. Su popularidad hizo que dicho plugin estuviera incorporado de base.

  Actualmente solo algunas páginas utilizan esta tecnología, por las quejas sobre su rendimiento y, sobre todo, por las capacidades que actualmente poseen los lenguajes HTML, CSS y JavaScript.

- **Applets**. Es otra tecnología prácticamente en extinción. Se trataba de componentes creados en leguaje Java que aprovechaban la potencia de este lenguaje para dotar de capacidades avanzadas a las aplicaciones web. Como en el caso de Flash, requiere de un plugin especial. La potencia actual del lenguaje JavaScript ha relegado completamente a esta tecnología, dejando su uso solo para aplicaciones muy concretas (y en muchos casos obsoletas).

Actualmente se habla de aplicaciones simplemente $HTML_5$, porque la última versión de la norma es la 5, y se da por hecho que $HTML_5$ incluye HTML, CSS y JavaScript. Esto simplifica mucho lo que un programador de aplicación web (en el lado del cliente) debe aprender.

Los lenguajes se van modernizando y adoptando nuevas funcionalidades que los navegadores van incorporando. Se exige a los navegadores que tengan capacidad, por ejemplo, de mostrar imágenes en varios

formatos, reproducir vídeo, mostrar notificaciones al usuario e incluso manejar periféricos como cámaras y micrófonos.

En definitiva, el navegador se ha convertido en un software que simula ser toda una máquina de trabajo comparables a las aplicaciones clásicas de escritorio.

## 2.2. EL TRIUNFO DE JAVASCRIPT

### 2.2.1. INICIOS DE JAVASCRIPT

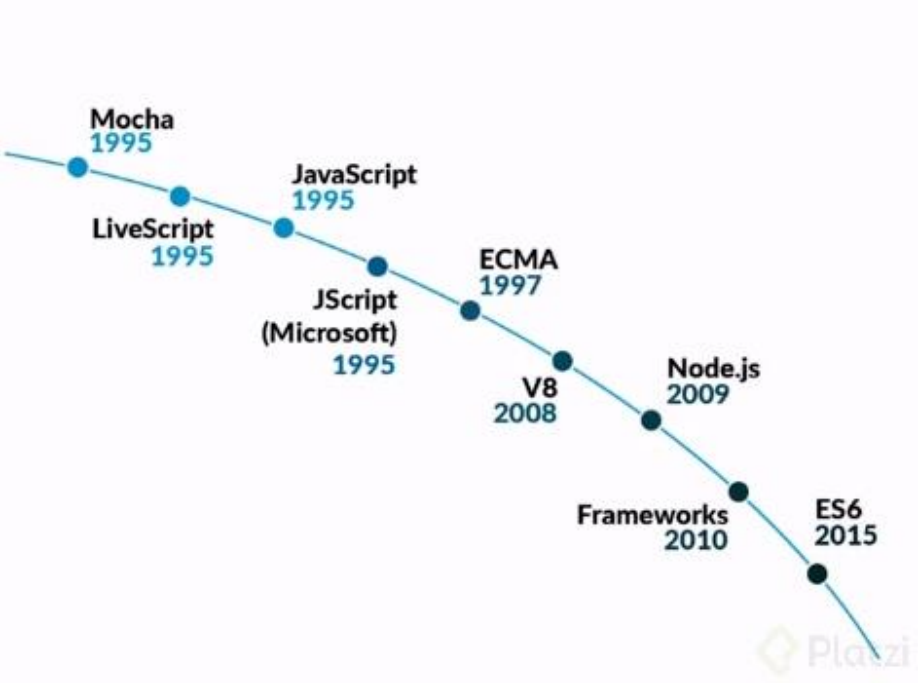
El desarrollo de aplicaciones web ha dispuesto de varias tecnologías front-end a lo largo de la historia. Pero lo cierto es que, por encima de todas, ha destacado JavaScript hasta ser el único lenguaje de programación que los navegadores son capaces de traducir sin necesidad de plugins o extensiones.

Inicialmente JavaScript se ideó para dotar de dinamismo a las páginas web. Las primeras páginas web se hacían en HTML y, aunque se añadió las primeras versiones de CSS, lo que las páginas web podían hacer era todavía muy poco.

Consciente del problema, la empresa **Netscape** decidió crear un lenguaje para que su navegador **Navigator** pudiera dar dinamismo a las páginas web. Inicialmente su nombre fue **LiveScript** (anteriormente **Mocha**).

El lenguaje **Java,** que en esa misma época tuvo un gran éxito, era propiedad de ***Sun Microsystems*** (poseía los derechos de explotación de la marca **Java**.) ***Netscape*** pidió permiso para utilizar la marca JavaScript a ***Sun Microsystems***, (cambiando el nombre de **LiveScript** por **JavaScript**, aunque realmente era muy diferente del lenguaje Java). ***Sun Microsystems*** se hizo propietaria de la marca *JavaScript*, pero permitió su uso a ***Netscape***.

***JavaScript*** tuvo un éxito inmediato y las compañías que fabricaban otros navegadores adoptaron el lenguaje. Microsoft desarrolló su propia versión a la que llamó **JScript** y otras empresas crearon otros (como **ActionScript** creado por Macromedia). El hecho de que los navegadores *Microsoft Internet Explorer* y *Netscape Navigator* coparan el mercado hizo que ambas versiones, que tenían bastantes diferencias, compitieran entre sí creado un problema a los desarrolladores de aplicaciones web.




## 2.2.2. NORMALIZACIÓN DE JAVASCRIPT

Durante mucho tiempo la tecnología de creación de páginas web se llamó **DHTML** (**Dynamic HTML**) porque con JavaScript algunas páginas eran ya dinámicas y empezaban a reaccionar a las acciones del usuario. Pero en estos años en los que se desarrolló la llamada Guerra de navegadores entre **Microsoft** y **Netscape** (que finalmente ganó Microsoft) se agravó el problema de las diferentes versiones de JavaScript.

Por ello se empezó a tratar la cuestión de estandarizar el lenguaje. La **World Wide Consortium,** organismo de estándares para las tecnologías relacionadas con HML y, sobre todo **ECMA**, organismo de estándares para la comunicación y la información decidieron estandarizar el lenguaje. ECMA lo normalizó por primera vez en 1997, llamándole **ECMAScript**.

Actualmente el nombre **JavaScript**, como marca registrada, es propiedad de ***Oracle Corporation*** al adquirir la empresa ***Sun Microsystems***. ***Netscape*** se disolvió a principios de siglo y dejó el proyecto como proyecto de código abierto impulsado por la ***Fundación Mozilla***. Por ello, esta fundación obtuvo el derecho de usar también el nombre de **JavaScript**.

Cuando nos referimos a **ECMAScript** hablamos del estándar actual de **JavaScript**.

La versión 5 del estándar (ECMAScript $_5$) fue totalmente adoptada por todos los navegadores en torno al año 2012, lo que mitigó el problema de las diferentes versiones de cada navegador. ECMA ha seguido publicando estándares que, poco a poco, la industria va adoptando.

## 2.2.3. APLICACIONES ENRIQUECIDAS

Google desarrolló el navegador **Chrome** y consiguió crear un motor, llamado **V8**, capaz de ejecutar JavaScript a una gran velocidad. El crecimiento continuo de JavaScript hizo que aparecieran aplicaciones web que ofrecían una UX semejante a la de las aplicaciones de escritorio. Se las llamó **RIA** (**Rich Internet Applications**) y en su potencia fue fundamental el aumento de las capacidades y la velocidad del lenguaje JavaScript.

Se vieron favorecidas con el desarrollo de HTML5, estándar actual de la tecnología front-end que incluyó la tercera versión de CSS junto con la modernización del lenguaje HTML (en detrimento del XHTML), además del impulso definitivo de JavaScript como único lenguaje de ejecución en el lado del navegador.

### 2.2.4. JAVASCRIPT SALE DEL NAVEGADOR

Todo ello ha contribuido a desarrollar aún más JavaScript y a tener un éxito absoluto como pieza central del desarrollo de aplicaciones web. Así en el año 2009, Ryan Dahl desarrolló un software llamado **Node.js** o **Node**.

Puesto que el motor V8 de Google que servía para traducir JavaScript en Chrome era un proyecto de código abierto, Dahl decidió utilizar el código para crear un entorno independiente del navegador, capaz de interpretar código JavaScript. Esto ha permitido, entre otras posibilidades, crear aplicaciones JavaScript en el lado del servidor.

Fue otro éxito inmediato que ha contribuido a ver a JavaScript como un lenguaje completo capaz de crear aplicaciones de todo tipo y que podía rivalizar con los lenguajes clásicos de programación.

El impulso dado por Node.js con ayuda de las múltiples librerías y frameworks de las que se puede dotar ha permitido a JavaScript ser un lenguaje capaz de:

- Crear aplicaciones de escritorio con ayuda de frameworks como **Electron** o **NW.js**.
- Programar dispositivos hardware con ayuda de frameworks como **Johnny-Five**, que tiene mucho éxito para programar placas **Arduino**, **Cylon.js** o **Node-Red**.
- Generar aplicaciones móviles nativas. Las aplicaciones móviles se crean usando lenguajes específicos: por ejemplo, **Java**, en dispositivos Android, y **Swift** u **ObjectiveC** en dispositivos iOS. Hoy en día es posible programas una aplicación HTML5 usando frameworks de JavaScript que permiten manipular el hardware del dispositivo. Algunos frameworks populares de este tipo son: **PhoneGap/Cordova, Ionic, Flutter** o **React Native**.
- Ser el lenguaje de manipulación de SGBD, como por ejemplo con MongoDB.

En definitiva, el salto de JavaScript para ser un lenguaje multipropósito, más allá de ser el lenguaje fundamental de creación de páginas web, es total.

### 2.2.5. JSON

**JSON** (**JavaScript Object Notation**) es un formato de documento, procedente de forma en la que se definen los objetos en JavaScript. Este, es un lenguaje capaz de manipular objetos y se diseñó en el lenguaje una forma particular de definir objetos.

La popularidad de este formato ha sido tal que actualmente ha traspasado al propio lenguaje y se le considera un formato de intercambio de datos independiente de JavaScript. Es decir, es una forma de definir datos que ha resultado tan potente y eficiente que ahora muchos lenguajes son capaces de leer y manipular datos en formato JSON de manera nativa, dejando completamente arrinconado al lenguaje XML como preferente para el intercambio de datos.

# 1. VERSIONES DE JAVASCRIPT

## 3.1. LA COMPATIBILIDAD

A medida que ha ido creciendo el uso del lenguaje, se le han ido exigiendo más funcionalidades y, sobre todo, más capacidades para ser usado como lenguaje completo de trabajo.

Las mejoras al lenguaje las incorporaban las empresas sin consensuar en absoluto dichas mejoras. La razón era que los fabricantes de navegadores dispusieran de un lenguaje mejor que el de sus rivales.

Sin embargo, la estandarización del lenguaje comenzó desde el momento que ***Netscape*** envió el borrador de JavaScript a ECMA para ello. La norma **ECMA-262** es la que se relaciona con dicha estandarización, llamando al lenguaje estandarizado **ECMAScript**.

Pero, aunque Microsoft y Netscape se comprometieron con el estándar, lo cierto es que **JavaScript** y **JScript** continuaron añadiendo especificaciones al lenguaje.

Todo cambió con la versión 5 del lenguaje ECMAScript que apareció en el año 2009 con la aceptación de $HTML_5$ como estándar. Esta versión y en especial su sucesora, la 5.1. del año 2011, fue adoptándose por todos los navegadores y finalmente Microsoft respetó el estándar en las últimas versiones de IE y más aún con la aparición de Edge.

Cuando en 2014 la W3C publicó el estándar HTML5, también JavaScript en su versión 5.1 fue adoptado masivamente. El impulso definitivo como estándar lo dio en 2011 la agencia mundial de estándares **ISO** junto con **IEC** (Comisión de Electrotecnia Internacional) normalizando el lenguaje en la norma **ISO/IEC 16262:2011.**

## 3.2. LENGUAJES PREPROCESADOS

Las necesidades de los programadores empezaron a influir en empresas y otras entidades para crear nuevos lenguajes que mejoraran el lenguaje original.

El primero en conseguir popularidad fue **CoffeeScript**. Permitió dotar a JavaScript de una sintaxis inspirada en el lenguaje Python para poder escribir JavaScript de forma más rápida y eficaz. Actualmente está en retroceso, pero durante muchos años fue utilizado por miles de programadores.

*CoffeeScript* no respeta los estándares porque, de hecho, es un lenguaje nuevo. Eso significa que ningún navegador, de forma nativa, es capaz de interpretar código CoffeeScript. Por ello el código debe ser convertido a JavaScript.


Funcionamiento de un preprocesador JavaScript

Esta es la idea de un lenguaje preprocesado, se escribe código en un lenguaje y se convierte con ayuda de un software especial (preprocesador) a JavaScript estándar.

Bajo esa misma idea, en el año 2012 apareció **TypeScript**, lenguaje creado por Microsoft que amplía JavaScript para que reconozca, entre otras muchas cosas, tipos de datos avanzados y las ventajas de los lenguajes fuertemente tipados. Ha tenido mucho éxito y es todavía muy utilizado, más aún al ser el lenguaje base del exitoso framework **Angular** de Google.

La misma idea está detrás del lenguaje **Dart** desarrollado por Google para modernizar JavaScript, **Elm** que permite programar en un lenguaje puramente funcional o los intentos de **ClojureScript** o **Scala.js** para convertir código en los lenguajes **Clojure** y **Scala**, respectivamente, en código JavaScript.

## 3.3. ESTÁNDARES

La influencia de los lenguajes preprocesados para JavaScript y las claras necesidades de mejorar el lenguaje, dieron lugar a nuevos estándares.

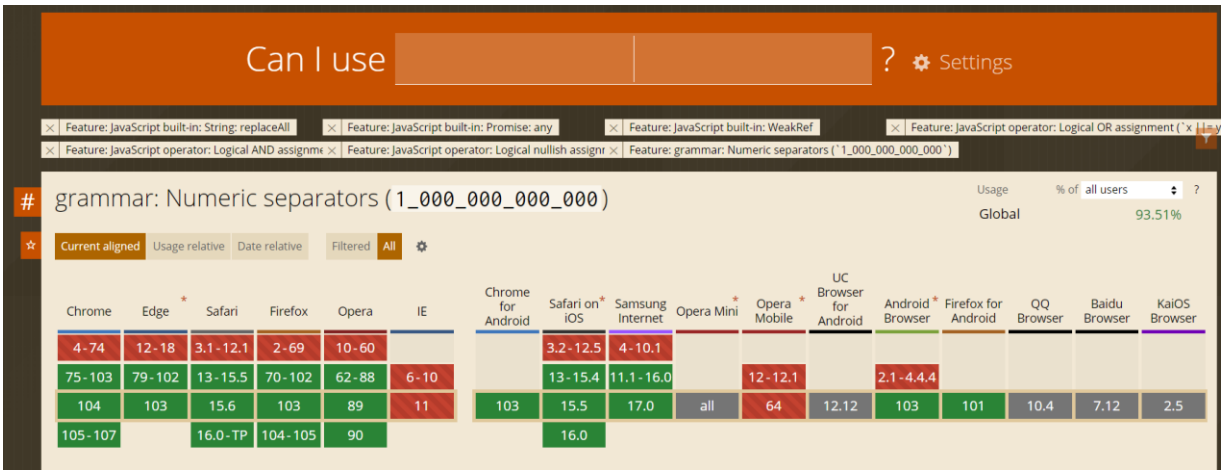
En el año 2015 apareció la norma **ECMAScript 6** (ES6 o ECMAScript 2015) que mejoró mucho el lenguaje JavaScript al dotarlo de elementos avanzados de otros lenguajes.

Han aparecido las versiones ES7 o ECMAScript 2016, ES8 o ECMAScript 2017, ES9 o ECMAScript 2018 ... hasta la actual ES13 o ECMAScript 2022. Y se esperan más.

La idea, al igual que ocurre con la propia norma $HTML_5$, es que los estándares sean vivos y sigan incorporando mejoras al lenguaje de forma veloz.

## 3.4. SITUACIÓN ACTUAL

Ante tanta variedad de lenguajes y normas, qué lenguaje aprender. Puede que la respuesta la disponga la página http://caniuse.com




Implantación del estándar ES2021, según la página CaIUse

Esta página nos permite saber hasta qué punto un aspecto de JavaScript (o de CSS) está implementado en los navegadores. Hoy en días es exigible que los usuarios tengan el navegador actualizado, pero debemos saber qué navegadores son compatibles con nuestro código y esa página es una clara ayuda.

La otra duda está en si utilizar lenguajes preprocesados u otras tecnologías facilitadoras, como **jQuery**. Lo aconsejable es aprender el lenguaje estándar para conseguir los mejores fundamentos posibles. Cuando se consigan esas bases, es verdad que hoy en día un programador front-end debe adquirir más habilidades que le permitan trabajar de forma rápida, eficiente y mantenible.