

Parte 1

El lenguaje JavaScript

JS

Ilya Kantor

Hecho el 21 de diciembre de 2025

La última versión de este tutorial está en <https://es.javascript.info>.

Trabajamos constantemente para mejorar el tutorial. Si encuentra algún error, por favor escríbanos a [nuestro github](#).

- [Una introducción](#)
 - [Una introducción a JavaScript](#)
 - [Manuales y especificaciones](#)
 - [Editores de Código](#)
 - [Consola de desarrollador](#)

Aquí aprenderemos JavaScript, empezando desde cero y llegaremos hasta conceptos avanzados como POO.

Nos concentraremos en el lenguaje mismo con el mínimo de notas específicas del entorno.

Una introducción

Acerca del lenguaje JavaScript y el entorno para desarrollar con él.

Una introducción a JavaScript

Veamos qué tiene de especial JavaScript, qué podemos lograr con este lenguaje y qué otras tecnologías se integran bien con él.

¿Qué es JavaScript?

JavaScript fue creado para “dar vida a las páginas web”.

Los programas en este lenguaje se llaman *scripts*. Se pueden escribir directamente en el HTML de una página web y ejecutarse automáticamente a medida que se carga la página.

Los scripts se proporcionan y ejecutan como texto plano. No necesitan preparación especial o compilación para correr.

En este aspecto, JavaScript es muy diferente a otro lenguaje llamado [Java](#) .

¿Por qué se llama JavaScript?

Cuando JavaScript fue creado, inicialmente tenía otro nombre: “LiveScript”. Pero Java era muy popular en ese momento, así que se decidió que el posicionamiento de un nuevo lenguaje como un “Hermano menor” de Java ayudaría.

Pero a medida que evolucionaba, JavaScript se convirtió en un lenguaje completamente independiente con su propia especificación llamada [ECMAScript](#) , y ahora no tiene ninguna relación con Java.

Hoy, JavaScript puede ejecutarse no solo en los navegadores, sino también en servidores o incluso en cualquier dispositivo que cuente con un programa especial llamado [El motor o intérprete de JavaScript](#) .

El navegador tiene un motor embebido a veces llamado una “Máquina virtual de JavaScript”.

Diferentes motores tienen diferentes “nombres en clave”. Por ejemplo:

- [V8](#) – en Chrome, Opera y Edge.
- [SpiderMonkey](#) – en Firefox.
- ...Existen otros nombres en clave como “Chakra” para IE , “JavaScriptCore”, “Nitro” y “SquirrelFish” para Safari, etc.

Es bueno recordar estos términos porque son usados en artículos para desarrolladores en internet. También los usaremos. Por ejemplo, si “la característica X es soportada por V8”, entonces probablemente funciona en Chrome, Opera y Edge.

¿Como trabajan los motores?


Los motores son complicados, pero los fundamentos son fáciles.

1. El motor (embebido si es un navegador) lee (“analiza”) el script.
2. Luego convierte (“compila”) el script a lenguaje de máquina.
3. Por último, el código máquina se ejecuta, muy rápido.

El motor aplica optimizaciones en cada paso del proceso. Incluso observa como el script compilado se ejecuta, analiza los datos que fluyen a través de él y aplica optimizaciones al código máquina basadas en ese conocimiento.



¿Qué puede hacer JavaScript en el navegador?

El JavaScript moderno es un lenguaje de programación “seguro”. No proporciona acceso de bajo nivel a la memoria ni a la CPU (UCP); ya que se creó inicialmente para los navegadores, los cuales no lo requieren.

Las capacidades de JavaScript dependen en gran medida en el entorno en que se ejecuta. Por ejemplo, [Node.JS](#)  soporta funciones que permiten a JavaScript leer y escribir archivos arbitrariamente, realizar solicitudes de red, etc.

En el navegador JavaScript puede realizar cualquier cosa relacionada con la manipulación de una página web, interacción con el usuario y el servidor web.

Por ejemplo, en el navegador JavaScript es capaz de:

- Agregar nuevo HTML a la página, cambiar el contenido existente y modificar estilos.
- Reaccionar a las acciones del usuario, ejecutarse con los clics del ratón, movimientos del puntero y al oprimir teclas.
- Enviar solicitudes de red a servidores remotos, descargar y cargar archivos (Tecnologías llamadas [AJAX](#)  y [COMET](#) ).
- Obtener y configurar cookies, hacer preguntas al visitante y mostrar mensajes.
- Recordar datos en el lado del cliente con el almacenamiento local (“local storage”).


¿Qué NO PUEDE hacer JavaScript en el navegador?

Las capacidades de JavaScript en el navegador están limitadas para proteger la seguridad de usuario. El objetivo es evitar que una página maliciosa acceda a información privada o dañe los datos de usuario.

Ejemplos de tales restricciones incluyen:

- JavaScript en el navegador no puede leer y escribir arbitrariamente archivos en el disco duro, copiarlos o ejecutar programas. No tiene acceso directo a funciones del Sistema operativo (OS).

Los navegadores más modernos le permiten trabajar con archivos, pero el acceso es limitado y solo permitido si el usuario realiza ciertas acciones, como “arrastrar” un archivo a la ventana del navegador o seleccionarlo por medio de una etiqueta `<input>`.

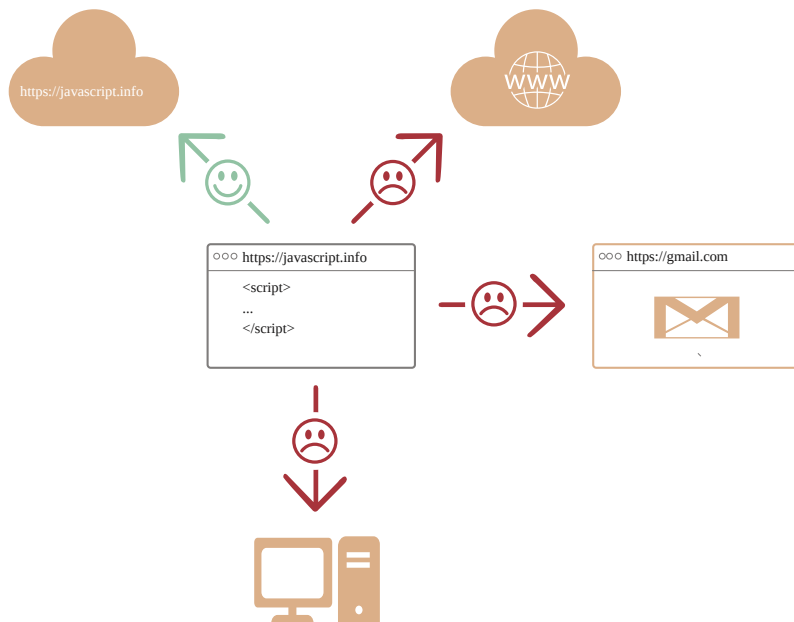
Existen maneras de interactuar con la cámara, micrófono y otros dispositivos, pero eso requiere el permiso explícito del usuario. Por lo tanto, una página habilitada para JavaScript no puede habilitar una cámara web para observar el entorno y enviar la información a la [NSA](#) .

- Diferentes pestañas y ventanas generalmente no se conocen entre sí. A veces sí lo hacen: por ejemplo, cuando una ventana usa JavaScript para abrir otra. Pero incluso en este caso, JavaScript no puede acceder a la otra si provienen de diferentes sitios (de diferente dominio, protocolo o puerto).

Esta restricción es conocida como “política del mismo origen” (“Same Origin Policy”). Es posible la comunicación, pero ambas páginas deben acordar el intercambio de datos y también deben contener el código especial de JavaScript que permite controlarlo. Cubriremos esto en el tutorial.

De nuevo: esta limitación es para la seguridad del usuario. Una página de `http://algunsitio.com`, que el usuario haya abierto, no debe ser capaz de acceder a otra pestaña del navegador con la URL `http://gmail.com` y robar la información de esta otra página.

- JavaScript puede fácilmente comunicarse a través de la red con el servidor de donde la página actual proviene. Pero su capacidad para recibir información de otros sitios y dominios está bloqueada. Aunque sea posible, esto requiere un acuerdo explícito (expresado en los encabezados HTTP) desde el sitio remoto. Una vez más: esto es una limitación de seguridad.



Tales limitaciones no existen si JavaScript es usado fuera del navegador; por ejemplo, en un servidor. Los navegadores modernos también permiten complementos y extensiones que pueden solicitar permisos extendidos.

¿Qué hace a JavaScript único?

Existen al menos *tres* cosas geniales sobre JavaScript:

- Completa integración con HTML y CSS.
- Las cosas simples se hacen de manera simple.
- Soportado por la mayoría de los navegadores y habilitado de forma predeterminada.

JavaScript es la única tecnología de los navegadores que combina estas tres cosas.

Eso es lo que hace a JavaScript único. Por esto es la herramienta mas extendida para crear interfaces de navegador.

Dicho esto, JavaScript también permite crear servidores, aplicaciones móviles, etc.

Lenguajes “por arriba de” JavaScript

La sintaxis de JavaScript no se adapta a las necesidades de todos. Personas diferentes querrán diferentes características.

Esto es algo obvio, porque los proyectos y requerimientos son diferentes para cada persona.

Así que recientemente han aparecido una gran cantidad de nuevos lenguajes, los cuales son *transpilados* (convertidos) a JavaScript antes de ser ejecutados en el navegador.

Las herramientas modernas hacen la conversión (Transpilación) muy rápida y transparente, permitiendo a los desarrolladores codificar en otros lenguajes y convertirlo automáticamente detrás de escena.

Ejemplos de tales lenguajes:

- [CoffeeScript](#) Es una “sintaxis azucarada” para JavaScript. Introduce una sintaxis corta, permitiéndonos escribir un código más claro y preciso. Usualmente desarrolladores de Ruby prefieren este lenguaje.
- [TypeScript](#) se concentra en agregar “tipado estricto” (“strict data typing”) para simplificar el desarrollo y soporte de sistemas complejos. Es desarrollado por Microsoft.
- [FLoow](#) también agrega la escritura de datos, pero de una manera diferente. Desarrollado por Facebook.

- [Dart](#) es un lenguaje independiente, tiene su propio motor que se ejecuta en entornos que no son de navegador (como aplicaciones móviles), pero que también se puede convertir/transpilar a JavaScript. Desarrollado por Google.
- [Brython](#) es un transpilador de Python a JavaScript que permite escribir aplicaciones en Python puro sin JavaScript.
- [Kotlin](#) es un lenguaje moderno, seguro y conciso que puede apuntar al navegador o a Node.

Hay más. Por supuesto, incluso si nosotros usamos alguno de estos lenguajes transpilados, deberíamos conocer también JavaScript para realmente entender qué estamos haciendo.

Resumen

- JavaScript fue inicialmente creado como un lenguaje solamente para el navegador, pero ahora es usado también en muchos otros entornos.
- Hoy en día, JavaScript tiene una posición única como el lenguaje más extendido y adoptado de navegador, con una integración completa con HTML y CSS.
- Existen muchos lenguajes que se convierten o transpilan a JavaScript y aportan ciertas características. Es recomendable echarles un vistazo, al menos brevemente, después de dominar JavaScript.

Manuales y especificaciones

Este libro es un *tutorial*. Su objetivo es ayudarte a aprender el lenguaje gradualmente. Pero una vez que te familiarices con lo básico, necesitarás otras fuentes.

Especificación

La [especificación ECMA-262](#) contiene la información más exhaustiva, detallada y formal sobre JavaScript. En ella se define el lenguaje.

Pero por su estilo formal, es difícil de entender a primeras. Así que si necesitas la fuente de información más fiable sobre los detalles del lenguaje, esta especificación es el lugar correcto a consultar. Es de entender entonces que no es para el uso diario.

Una nueva versión de la especificación del lenguaje es publicada anualmente. Entre publicaciones, el último borrador de la especificación se puede consultar en <https://tc39.es/ecma262/>.

Para leer acerca de las nuevas prestaciones de vanguardia del lenguaje, incluyendo aquellas que son “cuasi-estándar” (apodado “stage 3”), encuentra las propuestas en <https://github.com/tc39/proposals>.

Si estás desarrollando para navegadores web, se mencionan otras especificaciones en la [segunda parte](#) del tutorial.

Manuales

- **MDN (Mozilla) JavaScript Reference** es el manual principal, con ejemplos y otras informaciones. Es fantástico para obtener información exhaustiva sobre funciones individuales del lenguaje, métodos, etc.

Se puede acceder en <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>.

Aunque a menudo es preferible una búsqueda en internet. Simplemente añade “MDN [término]” en la consulta, por ejemplo <https://google.com/search?q=MDN+parseInt> para buscar la función `parseInt`.

Tablas de compatibilidad

JavaScript es un lenguaje en evolución, regularmente se agregan nuevas características.

Para ver la compatibilidad por navegador y otros motores, consultar:

- <https://caniuse.com> – tablas de compatibilidad por característica. Por ejemplo, para comprobar qué motores soportan funciones modernas de criptografía: <https://caniuse.com/#feat=cryptography>.

- <https://kangax.github.io/compat-table> – tabla que muestra la compatibilidad o no de las prestaciones del lenguaje por motor.

Todos estos recursos son de utilidad para el desarrollo con JavaScript, ya que incluyen información valiosa sobre los detalles del lenguaje, su compatibilidad, etc.

Por favor, tenlos en cuenta (o esta página) para cuando necesites información exhaustiva sobre una característica determinada.

Editores de Código

Un editor de código es el lugar donde los programadores pasan la mayor parte de su tiempo.

Hay dos principales tipos de editores de código: IDEs y editores livianos. Muchas personas usan una herramienta de cada tipo.

IDE

El término [IDE](#) (siglas en inglés para Integrated Development Environment, Ambiente Integrado de Desarrollo) se refiere a un poderoso editor con varias características que operan usualmente sobre un “proyecto completo”. Como el nombre sugiere, no sólo es un editor, sino un completo “ambiente de desarrollo”.

Un IDE carga el proyecto (el cual puede ser de varios archivos), permite navegar entre archivos, provee autocompletado basado en el proyecto completo (no sólo el archivo abierto), e integra un sistema de control de versiones (como [git](#)), un ambiente de pruebas, entre otras cosas a “nivel de proyecto”.

Si aún no has seleccionado un IDE, considera las siguientes opciones:

- [Visual Studio Code](#) (Multiplataforma, gratuito).
- [WebStorm](#) (Multiplataforma, de pago).

Para Windows, también está “Visual Studio”, no lo confundamos con “Visual Studio Code”. “Visual Studio” es un poderoso editor de pago sólo para Windows, idóneo para la plataforma .NET. Una versión gratuita es de este editor se llama [Visual Studio Community](#).

Muchos IDEs son de pago, pero tienen un periodo de prueba. Su costo usualmente es pequeño si lo comparamos al salario de un desarrollador cualificado, así que sólo escoge el mejor para ti.

Editores livianos

Los “editores livianos” no son tan poderosos como los IDEs, pero son rápidos, elegantes y simples.

Son usados principalmente para abrir y editar un archivo al instante.

La diferencia principal entre un “editor liviano” y un “IDE” es que un IDE trabaja a nivel de proyecto, por lo que carga mucha más información desde el inicio, analiza la estructura del proyecto si así lo requiere y continua. Un editor liviano es mucho más rápido si solo necesitamos un archivo.

En la práctica, los editores livianos pueden tener montones de plugins incluyendo analizadores de sintaxis a nivel de directorio y autocompletado, por lo que no hay un límite estricto entre un editor liviano y un IDE.

Existen muchas opciones, por ejemplo:

- [Sublime Text](#) (multiplataforma, shareware).
- [Notepad++](#) (Windows, gratuito).
- [Vim](#) y [Emacs](#) son también interesantes si sabes cómo usarlos.

No discutamos

Los editores en las listas anteriores son aquellos que yo o mis amigos a quienes considero buenos programadores hemos estado usando por un largo tiempo y con los que somos felices.

Existen otros grandes editores en este gran mundo. Por favor escoge el que más te guste.

La elección de un editor, como la de cualquier otra herramienta, es individual y depende de tus proyectos, hábitos y preferencias personales.

Opinión personal del author:

- Usaría [Visual Studio Code](#) si desarrollara mayormente “frontend”.
- De otro modo, si es mayormente otro lenguaje, plataforma, y solo parcialmente frontend; entonces consideraría otros editores, como XCode (Mac), Visual Studio (Windows) o la familia JetBrains (Webstorm, PHPStorm, RubyMine, etc.; dependiendo del lenguaje).

Consola de desarrollador

El código es propenso a errores. Es muy probable que cometas errores ... Oh, ¿de qué estoy hablando?

Definitivamente vas a cometer errores, al menos si eres un humano, no un [robot](#).

Pero el navegador, de forma predeterminada, no muestra los errores al usuario. Entonces si algo sale mal en el script, no veremos lo que está roto y no podemos arreglarlo.

Para ver los errores y obtener mucha otra información útil sobre los scripts, se han incorporado “herramientas de desarrollo” en los navegadores.

La mayoría de los desarrolladores se inclinan por Chrome o Firefox para el desarrollo porque esos navegadores tienen las mejores herramientas para desarrolladores. Otros navegadores también proporcionan herramientas de desarrollo, a veces con características especiales, pero generalmente están jugando a ponerse al día con Chrome o Firefox. Por lo tanto, la mayoría de los desarrolladores tienen un navegador “favorito” y cambian a otros si un problema es específico del navegador.

Las herramientas de desarrollo son potentes; Tienen muchas características. Para comenzar, aprenderemos cómo abrirlas, observar errores y ejecutar comandos JavaScript.

Google Chrome

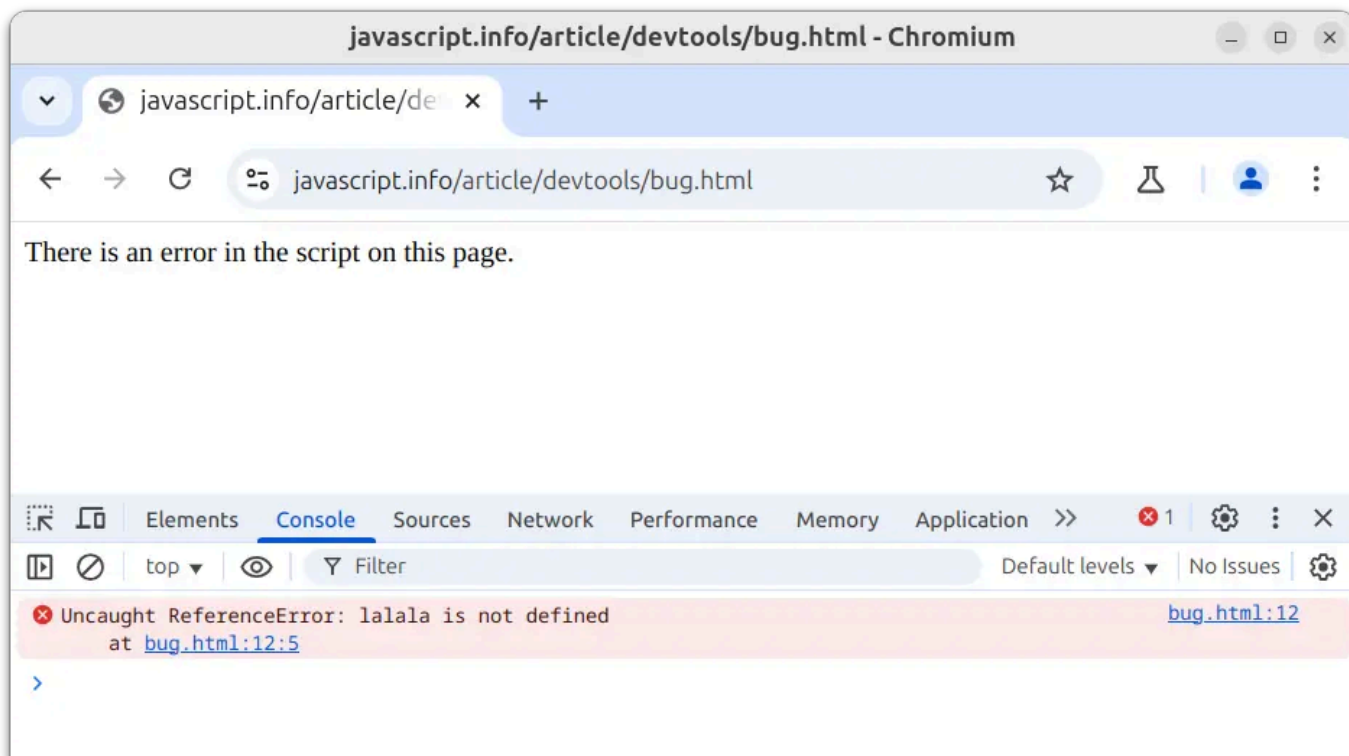
Abre la página [bug.html](#).

Hay un error en el código JavaScript dentro de la página. Está oculto a los ojos de un visitante regular, así que abramos las herramientas de desarrollador para verlo.

Presione **F12** o, si está en Mac, entonces combine **Cmd+Opt+J**.

Las herramientas de desarrollador se abrirán en la pestaña Consola de forma predeterminada.

Se ve algo así:



El aspecto exacto de las herramientas de desarrollador depende de su versión de Chrome. Cambia de vez en cuando, pero debería ser similar.

- Aquí podemos ver el mensaje de error de color rojo. En este caso, el script contiene un comando desconocido “lalala”.
- A la derecha, hay un enlace en el que se puede hacer clic en la fuente `bug.html:12` con el número de línea donde se produjo el error.

Debajo del mensaje de error, hay un símbolo azul `>`. Marca una “línea de comando” donde podemos escribir comandos JavaScript. Presione `Enter` para ejecutarlos.

Ahora podemos ver errores, y eso es suficiente para empezar. Volveremos a las herramientas de desarrollador más adelante y cubriremos la depuración más en profundidad en el capítulo [Debugging en el navegador](#).

i Entrada multilínea

Por lo general, cuando colocamos una línea de código en la consola y luego presionamos Enter, se ejecuta.

Para insertar varias líneas, presione `Shift+Enter`. De esta forma se pueden ingresar fragmentos largos de código JavaScript.

Firefox, Edge, y otros

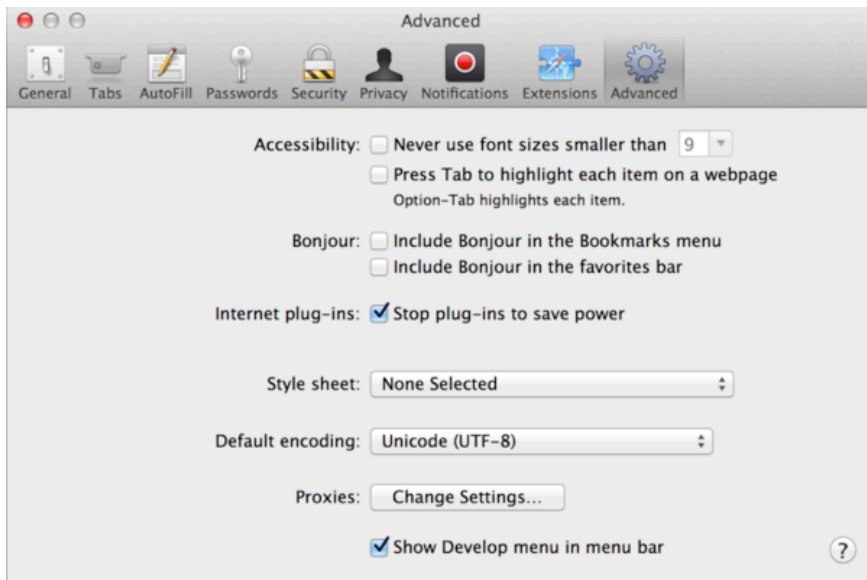
La mayoría de los otros navegadores usan `F12` para abrir herramientas de desarrollador.

La apariencia de ellos es bastante similar. Una vez que sepa cómo usar una de estas herramientas (puede comenzar con Chrome), puede cambiar fácilmente a otra.

Safari

Safari (navegador Mac, no compatible con Windows/Linux) es un poco especial aquí. Necesitamos habilitar primero el “Menú de desarrollo”.

Abra “Configuración” y vaya al panel “Avanzado”. Hay una casilla de verificación en la parte inferior:



Ahora combine `Cmd+Opt+C` para alternar a consola. Además, tenga en cuenta que ha aparecido el nuevo elemento del menú superior denominado “Desarrollar”. Tiene muchos comandos y opciones.

Resumen

- Las herramientas para desarrolladores nos permiten ver errores, ejecutar comandos, examinar variables y mucho más.
- Se pueden abrir con `F12` para la mayoría de los navegadores en Windows. Chrome para Mac necesita la combinación `Cmd+Opt+J`, Safari: `Cmd+Opt+C` (primero debe habilitarse).

Ahora tenemos el entorno listo. En la siguiente sección nos enfocaremos en JavaScript.