

(index.html)

UD1: Tecnologías, lenguajes y entorno de trabajo



Caso práctico

La empresa "BK Programación", ha sido informada a través de uno de sus asesores, de que se ha abierto el plazo, para concursar a la adjudicación de un proyecto de modernización de la web de una importante empresa dedicada a la moda.

Ada, la directora de "BK Programación", decide que es una buena oportunidad para conseguir trabajo para su empresa, por lo que consultan el pliego de requisitos exigidos y ve que su empresa tiene personal suficiente para dar solución a dicho proyecto. Se envía la solicitud y dos meses más tarde, la empresa de Ada sale adjudicataria del contrato de modernización.

El objetivo principal del proyecto es el de dar un mayor dinamismo a las páginas y actualizar la web a lo que se conoce como Web 2.0 (mayor interacción, interoperabilidad, aplicaciones más ricas y no intrusivas, etc.)

Ada considera que Antonio (estudiante de ciclo y trabajador de su empresa), con conocimientos de lenguaje HTML, podría formarse en las técnicas necesarias para realizar dicho trabajo de actualización y como además el proyecto tendrá un plazo máximo de entrega de 1 año, dispondrá de tiempo más que suficiente para ello.

Ada, informa a sus trabajadores de la adjudicación de dicho contrato, y Antonio (bajo la tutoría de Juan), decide comenzar a investigar en las diferentes opciones disponibles para llevar a cabo su trabajo.

En esta unidad didáctica nos vamos a introducir en los tipos de lenguajes de diseño web en entorno cliente, sus características, sus limitaciones, aspectos de seguridad y veremos algunas herramientas que podrás utilizar para configurar tu entorno de trabajo y comenzar a programar en JavaScript.

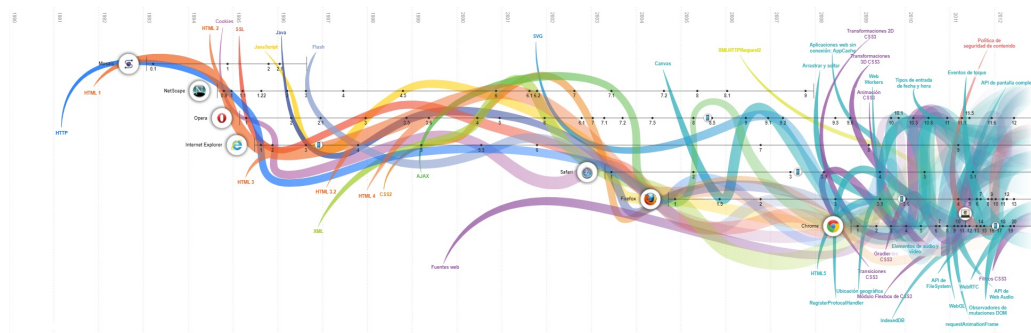
¡Pues vayamos a ello!

1.- Evolución de la web

La Web de hoy es un universo de aplicaciones y páginas web interconectadas lleno de vídeos, fotos y contenido interactivo. Lo que no ve el usuario es cómo interactúan los navegadores y las tecnologías web para hacer que esto sea posible.

A lo largo del tiempo, las tecnologías web han evolucionado hasta permitir que los desarrolladores puedan crear nuevas e increíbles experiencias web. La Web actual es el resultado de los continuos esfuerzos de una comunidad web abierta que ayuda a definir estas tecnologías web, tales como HTML5, CSS3 y WebGL, y garantiza que todos los navegadores web las admitan.

Las líneas de color de esta visualización representan la interacción entre los navegadores y las tecnologías web, lo que ha permitido el desarrollo del gran número de aplicaciones web increíbles que utilizamos a diario.



🔍 Pulsar para ampliar.



Para saber más

La web fue inicialmente concebida y creada por Tim Berners-Lee, un especialista del laboratorio europeo de partículas (CERN) en 1989. En sus mismas palabras, había una "necesidad de una herramienta colaborativa que soportara el conocimiento científico" en un contexto internacional. Él y su compañero Robert Cailliau crearon un prototipo web para el CERN y lo mostraron a la comunidad para sus pruebas y comentarios.

Dicho prototipo estaba basado en el concepto de hipertexto. Como resultado se crearon unos protocolos y especificaciones que han sido adoptados universalmente e incorporados a Internet, gracias a aportaciones posteriores como el desarrollo por la NCSA de la popular interfaz MOSAIC.

Todos los prototipos y desarrollos posteriores crecieron bajo la guía del consorcio W3C, que es una organización con base en el MIT de Massachusetts y que se responsabiliza de desarrollar y mantener los estándares web.

Por Web se pueden entender tres cosas distintas: el proyecto inicial del CERN, el conjunto de protocolos desarrollados en dicho proyecto o bien el espacio de información formado por todos los servidores interconectados. Cuando se habla de la Web generalmente se hace referencia a esto último.

Muchas de las discusiones sobre Diseño Web o Desarrollo Web son confusas, ya que la expresión varía considerablemente. Mientras que la mayoría de la gente tiene algún tipo de noción sobre lo que significa Diseño Web, solamente unos pocos son capaces de definirlo con exactitud, y tú vas a estar dentro de ese grupo.

Algunos componentes como diseño gráfico o programación, forman parte de esa discusión, pero su importancia en la construcción de webs varía de persona a persona y de web a web. Algunos consideran la creación y organización de contenido - o más formalmente, la arquitectura de la información - como el aspecto más importante del Diseño Web. Otros factores como - la facilidad de uso, el valor y funcionalidad del sitio web en la organización, su funcionalidad, accesibilidad, publicidad, etc. también forman una parte muy activa hoy en día sobre lo que se considera Diseño Web.

El Desarrollo Web ha sido y sigue estando muy influenciado por múltiples campos como el de las nuevas tecnologías, los avances científicos, el diseño gráfico, la programación, las redes, el diseño de interfaces de usuario, la usabilidad y una variedad de múltiples recursos. Por lo tanto el Desarrollo Web es realmente un campo multidisciplinar.

1.1.- Áreas

Hay cinco áreas que cubren la mayor parte de las facetas del Diseño Web:

- 1. Contenido:** incluye la forma y organización del contenido del sitio. Esto puede abarcar desde cómo se escribe el texto hasta cómo está organizado, presentado y estructurado usando tecnologías de marcas como HTML.
- 2. Visual:** hace referencia a la plantilla empleada en un sitio web. Esta plantilla generalmente se genera usando CSS y puede incluir elementos gráficos para decoración o para navegación. El aspecto visual es el aspecto más obvio del Diseño Web, pero no es la única disciplina o la más importante.
- 3. Tecnología:** aunque muchas de las tecnologías web como HTML o CSS entran dentro de esta categoría, la tecnología en este contexto generalmente hace referencia a los diferentes tipos de elementos interactivos de un sitio web, generalmente aquellos contruidos empleando técnicas de programación.
- 4. Distribución:** la velocidad y fiabilidad con la que un sitio web se distribuye en Internet o en una red interna corporativa está relacionado con el hardware/software utilizado y el tipo de arquitectura de red utilizada en la conexión.
- 5. Propósito:** la razón por la que un sitio web existe, generalmente está relacionada con algún aspecto de tipo económico. Por lo tanto este elemento debería considerarse en todas las decisiones que tomemos en las diferentes áreas.

El porcentaje de influencia de cada una de estas áreas en un sitio web, puede variar dependiendo del tipo de sitio que se está construyendo. Una página personal generalmente no tiene las consideraciones económicas que tendría una web que va a vender productos en Internet.

Una forma de pensar en los componentes del Diseño Web es a través de la metáfora de la pirámide mostrada en la figura anterior. El contenido proporciona los ladrillos que formarán la pirámide, pero la base de la pirámide se fundamenta tanto en la parte visual como en la parte tecnológica y con el punto de vista económico puesto como objetivo o propósito final en la mayoría de los casos.

Aunque la analogía de la pirámide es una forma un poco abstracta de describir el Diseño Web, es una herramienta que nos permite visualizar la interrelación de los diferentes componentes de la construcción Web.

Hoy en día los sitios web siguen un modelo basado en la programación cliente-servidor con tres elementos comunes:

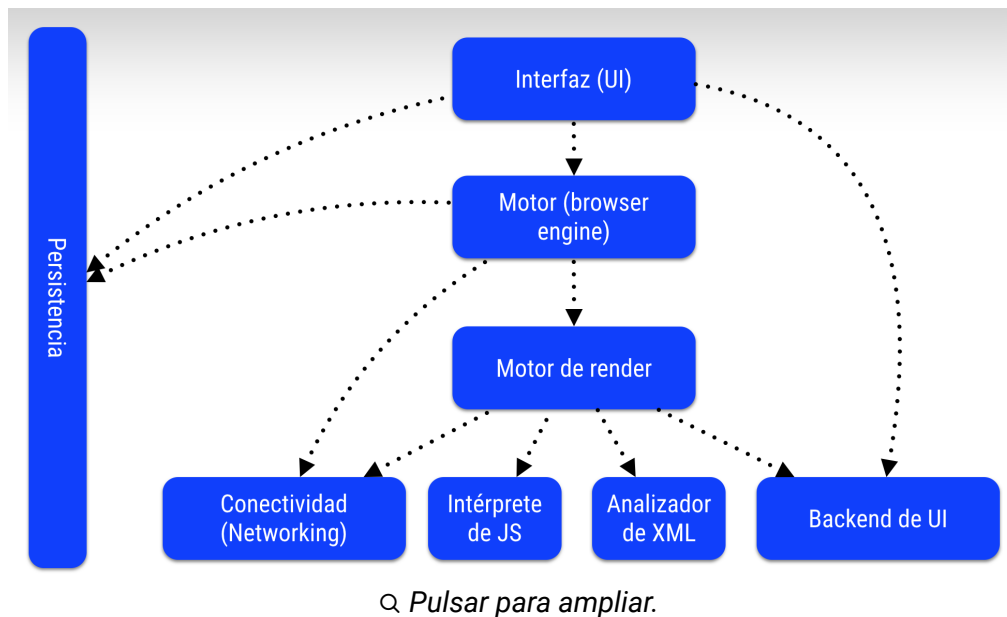
- El lado del servidor (server-side): incluye el hardware y software del servidor Web así como diferentes elementos de programación y tecnologías incrustadas. Las tecnologías pueden abarcar un rango amplio desde programas CGI escritos en PERL hasta aplicaciones multihilo basadas en Java, incluyendo tecnologías de servidor de bases de datos que soporten múltiples sitios web. Hoy día se utiliza mucho en el lado servidor el lenguaje node.js basado en JavaScript, pero para el lado del servidor
- El lado del cliente (client-side): este elemento hace referencia a los navegadores web y está soportado por tecnologías como HTML, CSS y lenguajes como JavaScript los cuales se utilizan para crear la presentación de la página o proporcionar características interactivas. Es justamente aquí dónde nos vamos a centrar a lo largo de todo el módulo.
- La red: describe los diferentes elementos de conectividad utilizados para mostrar el sitio web al usuario.

El entendimiento completo de todos los aspectos técnicos del medio Web, incluyendo la componente de red, es de vital importancia para llegar a ser un buen Diseñador Web.

2.- Arquitectura del navegador

Un navegador web es el encargado de renderizar e interpretar contenido en formato HTML, CSS, JS, XML y JSON que se encuentra almacenado de forma local o es recuperado de un servidor remoto (vía HTTP). Los navegadores también son capaces de pintar imágenes y archivos en diferentes formatos (dependiendo de si el browser tiene el plugin para el archivo respectivo). Independientemente del lenguaje usado del lado del servidor para la aplicación/página web, lo que un navegador entiende es HTML, CSS, JS, XML y JSON. Para esto, los navegadores modernos tienen diferentes capas de software. La siguiente imagen, basada en el artículo

[Browsers Work: Behind the scenes of modern web browsers](#), describe las capas:

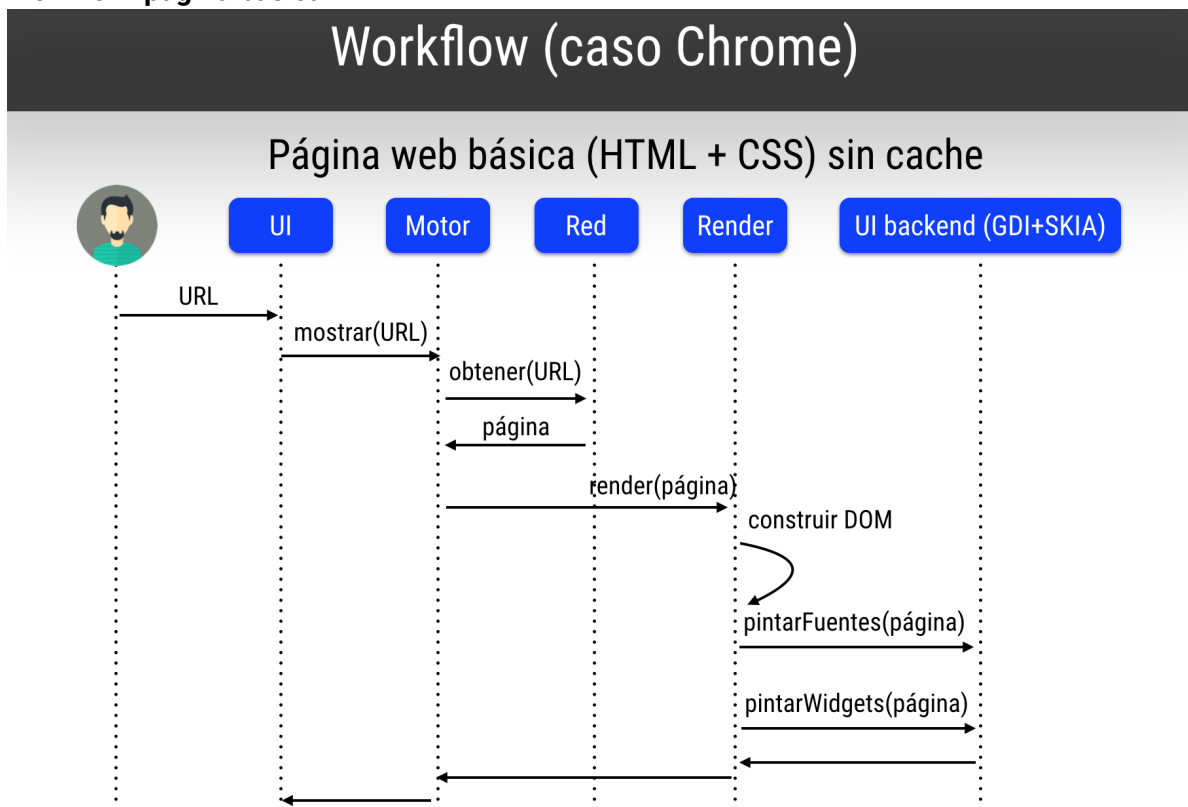


- **Interfaz (GUI):** es la capa que vemos los usuarios, donde ingresamos la URL a ver, definimos settings, abrimos pestañas, etc. Esta capa habla con las APIs locales de UI (es decir las del sistema operativo) para efectos de pintado de los widgets de la interfaz. Por ejemplo, la capa GUI de Firefox usa las librerías GTK (linux) o X11 (Unix) para pintar el browser y los componentes gráficos de las páginas.
- **Browser Engine:** es el controlador entre la vista y el motor de renderizado y otros componentes; orquesta las peticiones que hace el usuario con el resto de componentes en el browser.
- **Rendering Engine:** este es el corazón del browser. Se encarga de interpretar el código y pintarlo en la GUI. El motor de renderizado recibe código HTML y lo interpreta para construir el árbol de contenido (DOM) y el árbol de renderizado que luego se combinan para el pintado del contenido. Este motor también tiene un intérprete de CSS y para el caso de formatos especiales (ej., PDF) usa una arquitectura de plugins. Ejemplos de motores de renderizado que se usan en los browsers son: [Webkit](#) (Safari), [Blink](#) (Chromium, Opera), [Gecko](#) (Firefox), [Quantum](#) (Firefox), [EdgeHTML](#) (Microsoft Edge). En algunos casos el motor de browser y el de renderizado son el mismo, como en el caso de Gecko.
Intérprete de JS: fiel amigo del rendering engine que se encarga de interpretar y ejecutar código JS. Ejemplos de intérpretes de JS usados en browsers son: [SpiderMonkey](#) (Firefox), [V8](#) (Chrome, Android browser), Nitro (Safari), [JavaScriptCore](#) (Safari, Chrome para iOS).
- **Analizador de XML:** intérprete de XML (ej., Expat para Firefox).
- **Componente de conectividad:** se encarga de hacer las solicitudes usando el protocolo HTTP y el stack propio de cada sistema operativo.

- Componente de persistencia: capa de persistencia en el browser para almacenamiento local de datos como cookies. Dependiendo del browser, soporta otros mecanismos de almacenamiento como [localStorage](#) e [IndexedDB](#).
- Backend de UI: interfaz de comunicación con las librerías gráficas propias de cada sistema operativo.

Las siguientes diapositivas ilustran de forma general cómo se integran los componentes en casos de uso específicos:

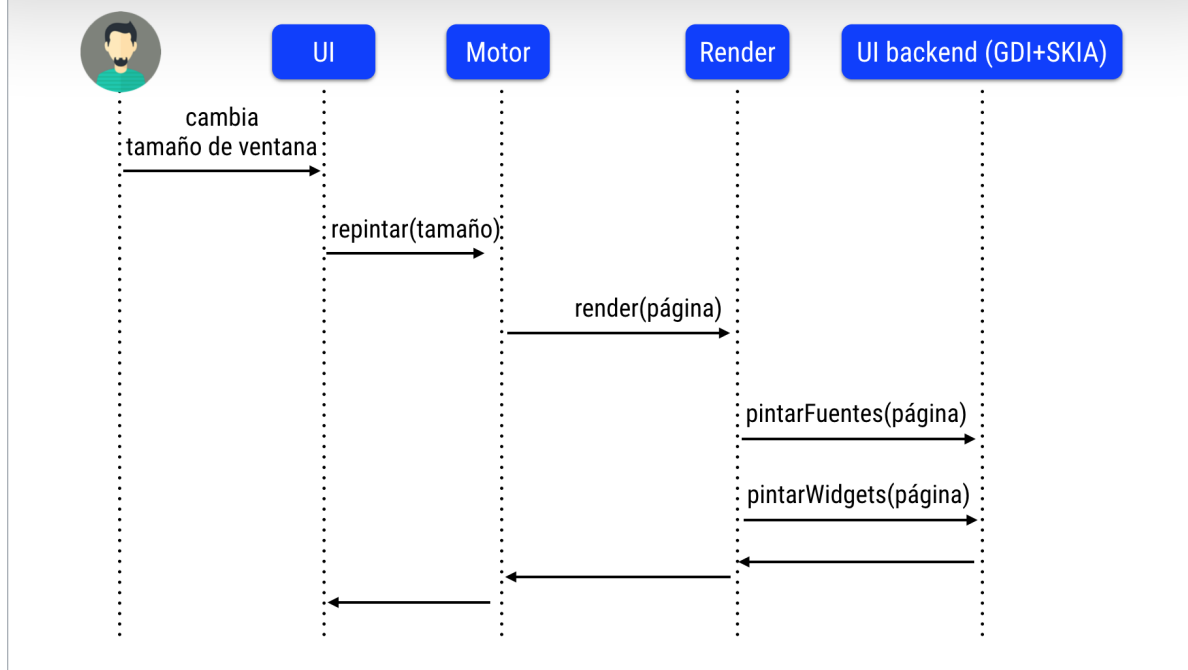
Workflow: página básica



Workflow: redimensionar navegador

Workflow (caso Chrome)

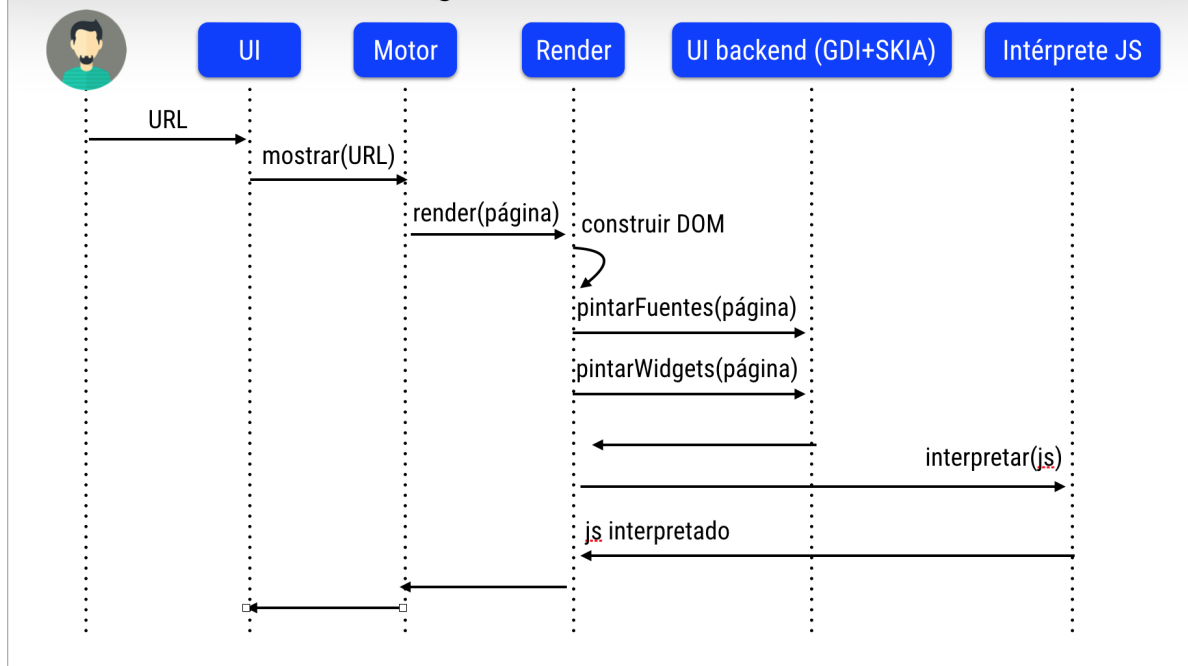
Cambiar dimensión de browser



Workflow: página en caché con js

Workflow (caso Chrome)

Página web en cache con JS



Para los interesados en entender más detalles de cómo funcionan los browsers, les recomiendo leer el artículo titulado [How Browsers Work: Behind the scenes of modern web browsers](#).

3.- Lenguajes de programación web

Existen multitud de lenguajes para programar web:

Java

Java es un lenguaje de programación open source y multiplataforma que, gracias a su versatilidad, es adecuado para, prácticamente, cualquier proyecto. Como la mayoría de los lenguajes web más conocidos, está orientado a objetos, es decir, depende de su campo de aplicación concreto. En internet existen incontables marcos y bibliotecas que están, generalmente, muy bien documentados, lo que facilita la ejecución de proyectos web, aunque sean muy complejos. Además, los programas escritos en Java son extensibles, escalables y fáciles de mantener siempre que el programador que esté realizando el proyecto sea un experto.

El hecho de que Java sea, en comparación con otros códigos, difícil de aprender hace que existan, como consecuencia, menos expertos de este lenguaje de programación web. Como clientes, esto nos supone esperar un precio más alto en relación con otros lenguajes de programación. De este modo, los programadores que dominan Java tienen mayores probabilidades de obtener ingresos relativamente altos.

JavaScript

El lenguaje de scripts dinámico orientado a objetos no guarda relación con Java a pesar de su nombre, aunque ambos comparten el hecho de estar escritos en C. Netscape desarrolló JavaScript por primera vez en 1995 con el nombre de LiveScript y el objetivo de extender HTML y CSS para que los programadores pudieran evaluar las interacciones de los usuarios y presentar el contenido de forma dinámica. Actualmente, JavaScript no se utiliza exclusivamente en navegadores web, sino también en microcontroladores y en servidores. El nombre JavaScript se eligió en base a la popularidad de Java, lenguaje al que se quería complementar. Fue todo un éxito: actualmente, las páginas web más conocidas utilizan, casi sin excepción, JavaScript como lenguaje de programación del lado del cliente. Además, existen muchos marcos y bibliotecas para JavaScript.

Este lenguaje de programación web presenta una escritura dinámica y no tiene clases. Por ello, los programadores pueden elegir entre programación orientada a objetos, de procedimiento o funcional, lo que aporta versatilidad a este lenguaje de programación. Esto se evidencia especialmente en los navegadores web: podrás, entre otras cosas, manipular dinámicamente el contenido de una página web, validar un formulario antes de enviarlo al servidor, activar cuadros de diálogo e integrar los scripts de carga y descarga. Además, JavaScript se ha convertido en una herramienta indispensable para programar cualquier web.

PHP

El preprocesador de hipertexto, más conocido por sus siglas PHP, es un lenguaje de scripting escrito en Perl y C. Se utiliza, principalmente, en la programación de páginas web y aplicaciones web dinámicas. PHP es considerado como un lenguaje de programación web apto para inexpertos y es compatible con HTML. Por estas razones, suele ser uno de los primeros lenguajes que aprenden los futuros programadores. A pesar de que hay quienes consideran que PHP es un lenguaje desactualizado, muchos propietarios de páginas web siguen dependiendo de él hoy en día. Entre las principales ventajas que ofrece, se incluye el hecho de que es un lenguaje con soporte de base de datos y una integración eficiente con el protocolo de internet. PHP ha publicado varias actualizaciones desde sus inicios y actualmente se encuentra en la versión 7. Se trata de un lenguaje con licencia de código abierto y disponible de forma gratuita.

PHP procesa el código del lado del servidor evitando así, la interpretación por parte del navegador, como ocurre en el caso de otros lenguajes de programación web muy conocidos. Por ello, PHP está incorporado en HTML (el cual no contiene información de estado) y provoca una mayor carga en el servidor que otros lenguajes de programación que solo transfieren el código fuente a un navegador web dedicado. Una crítica recurrente a PHP se basa en su escritura débil y en la falta de opciones dentro de la gestión de errores estandarizada, aunque el hecho de que muchas páginas web importantes sigan utilizando PHP es una prueba de la popularidad de este lenguaje. No obstante, pueden aparecer problemas si los programadores utilizan versiones de PHP no actualizadas que ponen en peligro la seguridad y la estabilidad de la página web.

Python

Python es un lenguaje de programación web de alto nivel basado en un código compacto, pero con una sintaxis fácil de entender. Python es también fácil de escribir porque, por ejemplo, los bloques no están separados con caracteres especiales, sino mediante sangrías. Y esta es precisamente la razón por la que este lenguaje es sencillo de aprender y de utilizar. Según cuáles sean tus necesidades, podrás utilizarlo para implementar una programación orientada a objetos, a aspectos o funcional. Además, Python es dinámico y se utiliza frecuentemente como lenguaje de scripting. El proyecto Python está impulsado por una comunidad activa que lo mantiene actualizado y conforme a los estándares de la industria a través de la fundación sin ánimo de lucro Python Software Foundation. Este lenguaje está disponible de forma gratuita y puede utilizarse en la mayoría de los sistemas operativos más conocidos.

Muchos de los servicios web más conocidos, como YouTube y otros proyectos del grupo Google, dependen parcialmente de Python. La industria de los videojuegos también ha descubierto y utiliza este lenguaje de programación. Lo mismo ocurre con los proyectos científicos, ámbito en el que Python goza de gran popularidad, principalmente, porque permite integrar fácilmente la mayoría de las bases de datos científicas y es eficiente en la resolución de tareas de recopilación de datos empíricos. Por estas razones, Python se considera un lenguaje de programación importante en estos campos, especialmente entre los principiantes. Para sus críticos, la velocidad de ejecución es relativamente baja y sus métodos tienen una definición un tanto engorrosa.

Ruby

Otro lenguaje de programación de alto nivel lo encontramos en el proyecto Ruby, desarrollado por Yukihiro Matsumoto a mediados de los años noventa. Este sencillo lenguaje de programación orientado a objetos ha convencido a muchos no solo por ofrecer una escritura dinámica y permitir reflexión sobre los objetos y las listas, sino también gracias a su recolector de basura automático. La principal característica distintiva de Ruby es el enfoque que tiene en el objeto: todo se considera objeto, tanto los valores como las clases. A diferencia de otros lenguajes de programación web orientados a objetos, Ruby no presenta excepciones aplicables a los tipos de datos primitivos. En resumen: “todo es objeto”.

En aras de la comodidad, la sintaxis de Ruby es flexible. Por ejemplo, el uso de paréntesis suele ser opcional. Esto lo convierte en un lenguaje muy fácil de leer que, en muchas ocasiones, parece a simple vista un lenguaje de marcado. No obstante, Ruby ofrece una gran potencia y permite también la metaprogramación, una habilidad que los desarrolladores utilizan para generar sus propios métodos, manipular la jerarquía de herencia y modificar otras constantes del lenguaje de programación y así poder personalizarlas. Por eso se dice que Ruby es “fácil de aprender, pero difícil de dominar”. Sus críticos llaman la atención sobre las consecuencias negativas que se derivan cuando hay errores tipográficos en el código: en esos casos, esos errores inesperados que afectan al tiempo de ejecución pueden acabar desencadenando desesperantes procesos de resolución de problemas. Con frecuencia, Ruby se utiliza como un lenguaje de scripting para servidores web, aunque es también increíblemente popular dentro de la industria de los videojuegos. Este lenguaje está disponible para los principales sistemas operativos.

C++

C++ está basado en C, uno de los lenguajes de programación más antiguos. Se empezó a desarrollar en 1979 y estaba pensado como una extensión de aquel. Hubo que esperar hasta 1985 para que fuera puesto a disposición del público. Hasta la fecha, es un lenguaje muy popular. C++ es un lenguaje de programación ratificado como estándar ISO (Organización Internacional de Normalización) que se considera tanto de bajo nivel y eficiente como complejo y con alta capacidad de abstracción. C++ es, en términos generales, fácil de aprender, sobre todo porque el núcleo del lenguaje es muy abarcable e incluye aproximadamente 60 palabras clave. El lenguaje se vuelve más complejo y gana en alcance gracias a su biblioteca estándar.

Las mayores fortalezas del lenguaje C++ son su gran variedad de combinaciones y su eficiente programación de bajo nivel. Es posible agrupar en funciones básicas incluso los procesos de mayor complejidad. Por eso, los programadores de C++ se ahorran mucho trabajo al poder confiar en el núcleo del lenguaje y en la biblioteca estándar. Debido a que se basa estrictamente en C, este lenguaje de programación tiene algunas desventajas, como, por ejemplo, una sintaxis desordenada en comparación con otros lenguajes. No obstante, C++ es actualmente uno de los lenguajes de programación más utilizados en el ámbito de la programación de aplicaciones y sistemas. Como lenguaje de programación web, C++ está por detrás de Java, JavaScript y C#.

C#

El relativamente joven lenguaje de programación C# ("C Sharp", en su voz inglesa), lanzado en 2001, se considera un lenguaje de propósito general. Sigue un sistema de tipos unificados, está orientado a objetos y es, en términos generales, multiplataforma, aunque al tratarse de un proyecto de Microsoft ha sido específicamente diseñado para .NET Framework. Es muy frecuente encontrarlo bajo el nombre "Visual C#", sobre todo como implementación. Conceptualmente, se trata de una evolución de Java y C++ que amplía el modelo orientado a objetos gracias a los llamados atributos, que almacenan información sobre clases, objetos y métodos, y a los delegados, que representan referencias a métodos determinados. Principalmente, esto nos permite conseguir una descripción de error más eficiente durante la compilación de código, algo que ahorra tiempo a los desarrolladores.

Para muchos, C# es, junto a Java, el lenguaje de programación más importante y que todo desarrollador web debería aprender. Como lenguaje de programación orientado a objetos, C# ofrece la mejor combinación entre funcionalidad y potencia. Sus críticos advierten del problema derivado de vincular el uso de C# a .NET Framework de Microsoft. No obstante, con esta sintaxis, los programadores cubren un gran sector del mercado: muchos confían actualmente en C# a la hora de programar para sistemas Windows o videojuegos para Xbox y PC. Como lenguaje de programación web, C# se utiliza principalmente en las API web y en varias aplicaciones web.

Perl

El lenguaje de programación gratuito Perl fue lanzado en 1987 como lenguaje de programación interpretado e inspiró, entre otros, los lenguajes PHP, JavaScript, Ruby y Python. Los desarrolladores se basaron, sobre todo, en los lenguajes de programación de la familia C. En términos generales, es un lenguaje multiplataforma, diseñado en principio para ser utilizado en la administración de redes y sistemas. Actualmente, Perl se ha establecido como uno de los lenguajes de programación más utilizados en el ámbito del software web, la bioinformática y las finanzas.

Con Perl, los programadores disfrutan de una gran libertad y de eficiencia en la resolución de problemas. Por ejemplo, los textos pueden ser editados con expresiones regulares y, además, existen muchos módulos gratuitos disponibles para Perl a los que se accede a través del módulo de biblioteca de Perl, CPAN. Como lenguaje de programación, Perl se mantiene fiel a sus principios de ofrecer siempre al programador varias formas de alcanzar su objetivo, continuar siendo sencillo y eficiente y actuar de manera sensible al contexto. Perl ha sido fundamental en la difusión de la World Wide Web y

sigue desempeñando un papel importante como lenguaje de programación web, aunque es cierto que se usa con menos frecuencia en ese sentido cuando la proximidad del hardware (por ejemplo, con los servidores web) y la velocidad (por ejemplo, de los controladores) son relevantes.

Estos son los lenguajes de programación web utilizados por las páginas web más conocidas. Muchas veces, al comenzar a programar, es común verse asediado por las numerosas opciones de lenguajes disponibles. Sin embargo, observando cómo funcionan las páginas web más importantes puede aprenderse: ¿Qué lenguajes de programación utilizan Facebook, Twitter o Google? ¿Cuáles son los mejores para el lado del cliente y cuáles para el lado del servidor?

En el siguiente resumen podemos ver que todas las páginas web mencionadas utilizan JavaScript del lado del cliente, pero dependen de una gran variedad de lenguajes de programación del lado del servidor.

Página web	Lenguaje de programación del lado del cliente	Lenguaje de programación del lado del servidor
Google	JavaScript	C, C++, Go, Java, Python, PHP (HHVM)
Facebook	JavaScript	Hack, PHP (HHVM), Python, C++, Java, Erlang, D, XHP, Haskell
YouTube	JavaScript	C, C++, Python, Java, Go
Yahoo	JavaScript	PHP
Amazon	JavaScript	Java, C++, Perl
Wikipedia	JavaScript	PHP, Hack
Twitter	JavaScript	C++, Java, Scala, Ruby

¿Viendo estos datos, adivinas cual es lenguaje más importante para programar en el lado cliente?

3.1.- JavaScript



Para saber más

¿Te gustaría ver cómo soporta cada navegador las diferentes características de JavaScript y poder verlo por versiones?

[Comparación de navegadores web.](#)

Características

Como vimos anteriormente, los lenguajes de programación para clientes web no son un reemplazo de la programación en el lado del servidor. Cualquier web que reaccione dinámicamente a interacciones del usuario o que almacene datos, estará gestionada por lenguajes de script en el lado del servidor, incluso aunque usemos JavaScript en el cliente para mejorar la experiencia de usuario. Las razones son simples:

- Primero: JavaScript por sí mismo no puede escribir ficheros en el servidor. Puede ayudar al usuario a elegir opciones o preparar datos para su envío, pero después de eso solamente podrá ceder los datos al lenguaje de servidor encargado de la actualización de datos.
- Segundo: no todos los clientes web ejecutan JavaScript. Algunos lectores, dispositivos móviles, buscadores, o navegadores instalados en ciertos contextos están entre aquellos que no pueden realizar llamadas a JavaScript, o que simplemente son incompatibles con el código de JavaScript que reciben. Aunque esto ocurra nuestra página web debería ser completamente funcional con JavaScript desactivado. Utilizaremos JavaScript para conseguir que la experiencia de navegación web sea lo más rápida, moderna o divertida posible, pero no dejaremos que nuestra web deje de funcionar si JavaScript no está funcionando.
- Tercero: uno de los caminos que más ha integrado la programación cliente con la programación servidor ha surgido gracias a AJAX. El proceso "asíncrono" de AJAX se ejecuta en el navegador del cliente y emplea JavaScript. Este proceso se encarga de solicitar datos XML, o enviar datos al lenguaje de servidor y todo ello de forma transparente en background. Los datos devueltos por el servidor pueden ser examinados por JavaScript en el lado del cliente, para actualizar secciones o partes de la página web. Es así como funcionan hoy día la mayoría de las web.

JavaScript está orientado a dar soluciones a:

- Conseguir que nuestra página web responda o reaccione directamente a la interacción del usuario con elementos de formulario y enlaces hipertexto.
- La distribución de pequeños grupos de datos y proporcionar una interfaz amigable para esos datos.
- Controlar múltiples ventanas o marcos de navegación, plug-ins, o applets Java basados en las elecciones que ha hecho el usuario en el documento HTML.
- Pre-procesar datos en el cliente antes de enviarlos al servidor.

- Modificar estilos y contenido en los navegadores de forma dinámica e instantáneamente, en respuesta a interacciones del usuario.
- Solicitar ficheros del servidor, y enviar solicitudes de lectura y escritura a los lenguajes de servidor.

Los lenguajes de script como JavaScript no se usan solamente en las páginas web. Los intérpretes de JavaScript están integrados en múltiples aplicaciones de uso cotidiano. Estas aplicaciones proporcionan su propio modelo de acceso y gestión de los módulos que componen la aplicación y para ello comparten el lenguaje JavaScript en cada aplicación.

Compatibilidades

A diferencia de otros tipos de scripts, JavaScript es interpretado por el cliente. Actualmente existen múltiples clientes o navegadores que soportan JavaScript, incluyendo Firefox, Google Chrome, Safari, Opera, Internet Explorer, etc. Por lo tanto, cuando escribimos un script en nuestra página web, tenemos que estar seguros de que será interpretado por diferentes navegadores y que aporte la misma funcionalidad y características en cada uno de ellos. Ésta es otra de las diferencias con los scripts de servidor en los que nosotros dispondremos del control total sobre su interpretación.

Cada tipo de navegador da soporte a diferentes características del JavaScript y además también añaden sus propios bugs o fallos. Algunos de estos fallos son específicos de la plataforma sobre la que se ejecuta ese navegador, mientras que otros son específicos del propio navegador en sí.

A veces las incompatibilidades entre navegadores al interpretar el código de JavaScript no vienen dadas por el propio código en sí, sino que su origen proviene del código fuente HTML. Por lo tanto es muy importante que tu código HTML siga las especificaciones del estándar W3C y para ello dispones de herramientas como el validador HTML W3C:

[Validador W3C](#).

También tienes que tener precaución con las limitaciones en el uso de JavaScript:

- No todos los navegadores soportan lenguajes de script (en especial JavaScript) en el lado del cliente.
- Algunos dispositivos móviles tampoco podrán ejecutar JavaScript.
- Incluso las implementaciones más importantes de JavaScript en los diferentes navegadores no son totalmente compatibles entre ellas: por ejemplo diferentes incompatibilidades entre Firefox e Internet Explorer.
- La ejecución de código JavaScript en el cliente podría ser desactivada por el usuario de forma manual, con lo que no podremos tener una confianza ciega en que se vaya a ejecutar siempre tu código de JavaScript.
- Algunos navegadores de voz, no interpretan el código de JavaScript.

Seguridad

JavaScript proporciona un gran potencial para diseñadores maliciosos que quieran distribuir sus scripts a través de la web. Para evitar esto los navegadores web en el cliente aplican dos tipos de restricciones:

- Por razones de seguridad cuando se ejecuta código de JavaScript éste lo hace en un "espacio seguro de ejecución" en el cuál solamente podrá realizar tareas relacionadas con la web, nada de tareas genéricas de programación como creación de ficheros, etc.
- Además los scripts están restringidos por la política de "mismo origen": la cuál quiere decir que los scripts de una web no tendrán acceso a información tal como usuarios, contraseñas, o cookies enviadas desde otra web. La mayor parte de los agujeros de seguridad son infracciones tanto de la política de "mismo origen" como de la política de "espacio seguro de ejecución".

Al mismo tiempo es importante entender las limitaciones que tiene JavaScript y que, en parte, refuerzan sus capacidades de seguridad. JavaScript no podrá realizar ninguna de las siguientes tareas:

- Modificar o acceder a las preferencias del navegador del cliente, las características de apariencia de la ventana principal de navegación, las capacidades de impresión, o a los botones de acciones del navegador.
- Lanzar la ejecución de una aplicación en el ordenador del cliente.
- Leer o escribir ficheros o directorios en el ordenador del cliente (con la excepción de las cookies).
- Escribir directamente ficheros en el servidor.
- Capturar los datos procedentes de una transmisión en streaming de un servidor, para su retransmisión.
- Enviar e-mails a nosotros mismos de forma invisible sobre los visitantes a nuestra página web (aunque si que podría enviar datos a una aplicación en el lado del servidor capaz de enviar correos).
- Interactuar directamente con los lenguajes de servidor.
- Las páginas web almacenadas en diferentes dominios no pueden ser accesibles por JavaScript.
- JavaScript es incapaz de proteger el origen de las imágenes de nuestra página.
- Implementar multiprocesamiento o multitarea.
- Otro tipo de vulnerabilidades que podemos encontrar están relacionadas con el XSS. Este tipo de vulnerabilidad viola la política de "mismo origen" y ocurre cuando un atacante es capaz de inyectar código malicioso en la página web presentada a su víctima. Este código malicioso puede provenir de la base de datos a la cuál está accediendo esa víctima. Generalmente este tipo de errores se deben a fallos de implementación de los programadores de navegadores web.

Otro aspecto muy relacionado con la seguridad son los defectos o imperfecciones de los navegadores web o plugins utilizados. Éstas imperfecciones pueden ser empleadas por los atacantes para escribir scripts maliciosos que se puedan ejecutar en el sistema operativo del usuario.

El motor de ejecución de JavaScript es el encargado de ejecutar el código de JavaScript en el navegador y por lo tanto es en él dónde recaerá el peso fuerte de la implementación de la seguridad. Podríamos citar varios ejemplos de motores de JavaScript:

- Active Script de Microsoft: tecnología que soporta JScript como lenguaje de scripting. A menudo se considera compatible con JavaScript, pero Microsoft emplea múltiples características que no siguen los estándares ECMA.

- El kit de herramientas Qt desarrollado en C++ también incluye un módulo intérprete de JavaScript.
- El lenguaje de programación Java en su versión JDK 1.6 introduce un paquete denominada javax.script que permite la ejecución de JavaScript.
- Y por supuesto todos los motores implementados por los navegadores web como Mozilla, Google, Opera, Safari, etc. Cada uno de ellos da soporte a alguna de las diferentes versiones de JavaScript.

Hoy en día una de las características que más se resalta y que permite diferenciar a unos navegadores de otros, es la rapidez con la que sus motores de JavaScript pueden ejecutar las aplicaciones, y la seguridad y aislamiento que ofrecen en la ejecución de las aplicaciones en diferentes ventanas o pestañas de navegación.

4.- Herramientas y utilidades



Caso práctico

En BK Programación Juan y Antonio han decidido, después de analizar los requisitos del proyecto y de consultarlo con su directora Ada, que utilizarán el lenguaje JavaScript para la realización del proyecto de modernización de la empresa de moda.

Lo primero que tienen que hacer es decidir qué tipo de herramientas y utilidades adicionales van a usar para realizar la programación con el lenguaje JavaScript. Buscan alguna herramienta que les permita introducir el código de JavaScript fácilmente y les aporte ayudas adicionales detectando errores sintácticos, partes incompletas, etc.

También tienen que decidir qué navegador o navegadores van a usar para comprobar la ejecución y compatibilidad de su código de JavaScript.

La mejor forma de aprender JavaScript es tecleando el código HTML y JavaScript en un simple documento de texto. La elección del editor depende de ti, pero aquí te voy a dar algunas pistas para realizar una buena elección.

Para aprender JavaScript no se recomiendan editores del estilo WYSIWYG, ya que estas herramientas están más orientadas a la modificación de contenido y presentación, y nosotros nos vamos a centrar más en el código fuente de la página.

Uno de los factores importantes que tienes que tener en cuenta a la hora de elegir un editor, es ver la facilidad con la que se pueden grabar los ficheros con extensión .html. Independientemente del sistema operativo que estés utilizando cualquier programa que te permita grabar ficheros directamente con dicha extensión te evitaría un gran número de problemas. También hay que tener en cuenta la codificación que emplea ese programa para grabar los ficheros.

Hoy día no es necesario contar con un IDE para desarrollar en Javascript. Hay editores ligeros que, a día de hoy, nos ofrecen casi las mismas características. Lo que sí debes tener en cuenta es que te aporten las siguientes características para hacer tu trabajo más fácil:

- Resaltado de texto. Muestra con distinto color o tipo de letra los diferentes elementos del lenguaje: sentencias, variables, comentarios, etc. También genera indentado automático para diferenciar de forma clara los distintos bloques de un programa.
- Completado automático. Detecta qué estás escribiendo y cuando es posible te muestra distintas opciones para completar el texto.
- Navegación en el código. Permite buscar de forma sencilla elementos dentro del texto, por ejemplo, definiciones de variables.
- Comprobación de errores al editar. Reconoce la sintaxis del lenguaje y revisa el código en busca de errores mientras lo escribes.
- Generación automática de código. Ciertas estructuras, como la que se utiliza para las clases, se repiten varias veces en un programa. La generación automática de código puede encargarse de crear la estructura básica, para que sólo tengas que rellenarla.
- Gestión de versiones. En conjunción con un sistema de control de versiones, el entorno de desarrollo te puede ayudar a guardar copias del estado del proyecto a lo largo del tiempo, para que si es necesario puedas revertir los cambios realizados.

Editores más populares para trabajar con JavaScript

VSCode

Visual Studio Code de Microsoft es una herramienta muy versátil, a la que se le pueden instalar muchas extensiones para darle una gran funcionalidad en casi cualquier lenguaje de programación. [Visual Studio Code](#).



Atom

Atom de GitHub es una herramienta también muy ligera y que también cuenta con multitud de extensiones para adaptarla a nuestras necesidades. [Atom](#).



Sublime Text

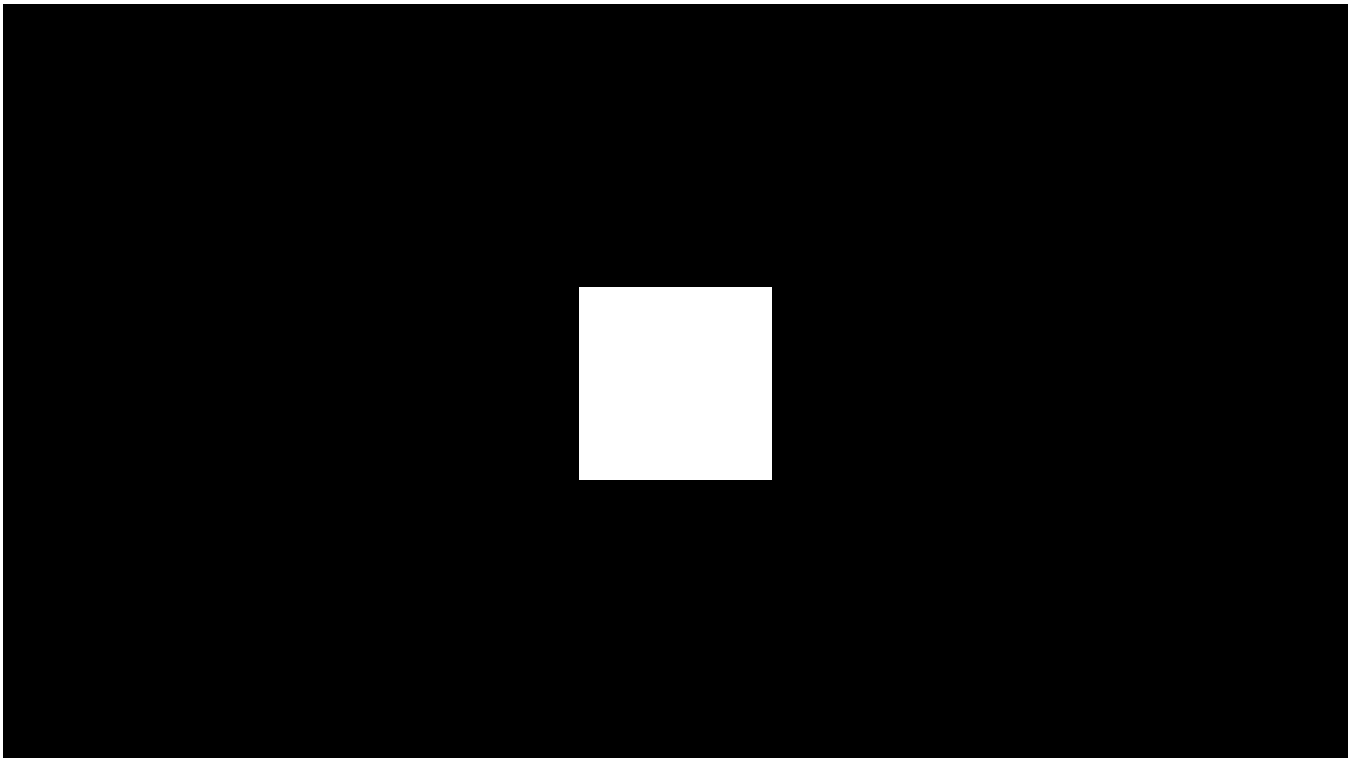
SublimeText al igual que los anteriores les pueden instalar gran cantidad de extensiones, aunque es software propietario pero se puede usar para enseñanza.



4.1- Instalación y configuración de Visual Studio Code

La elección de un editor u otro es algo que, en la actualidad, se ajusta más a los gustos y las necesidades de cada programador. No existe una herramienta que destaque sobre las demás en todos los aspectos. Mi propuesta que uses Visual Studio Code, ya que es un editor sencillo, con buena apariencia, ligero y muy configurable. Además, la mayoría de los desarrolladores de JavaScript lo usan de manera profesional y tienes documentación y tutoriales a raudales.

En este vídeo puedes ver el proceso de instalación de Visual Studio Code, te recomiendo que lo sigas e investigues un poco cómo configurarlo para que se ajuste poco a poco a tus gustos.



00:00

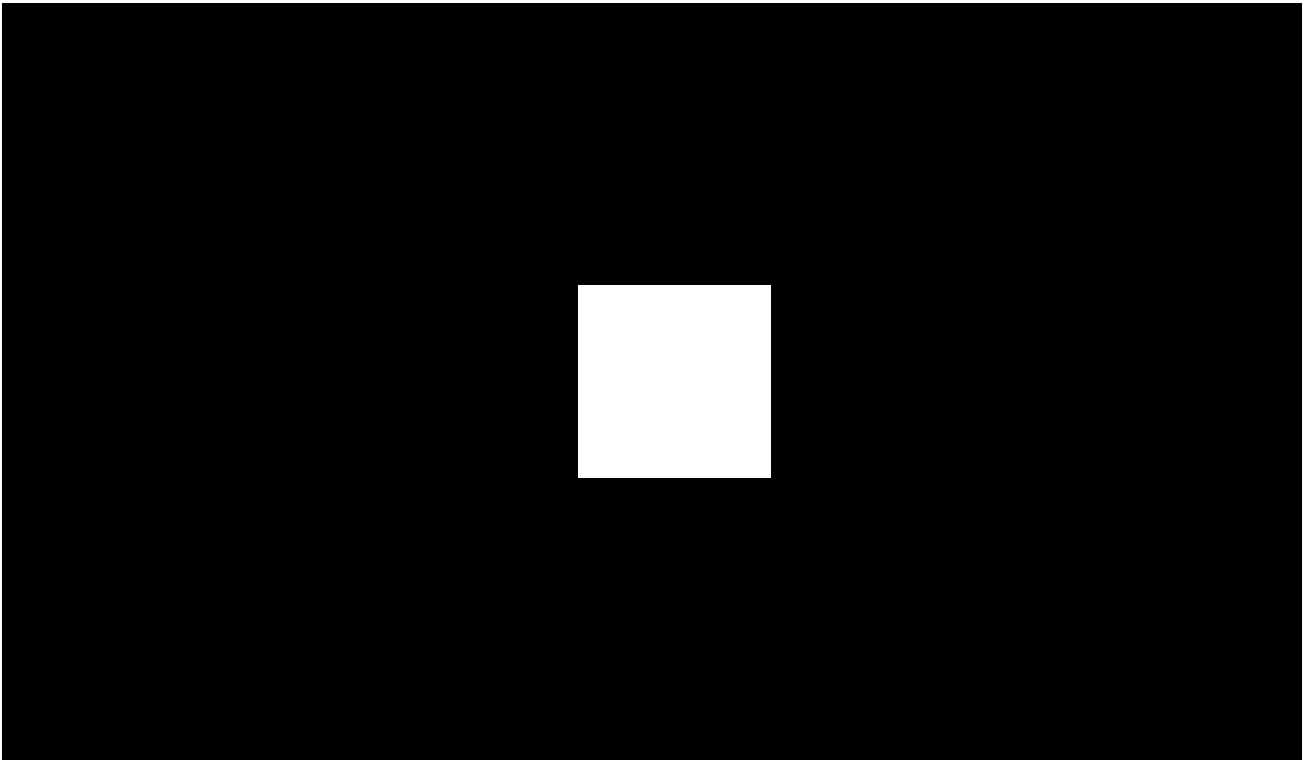
04:41

🔍 [Instalar y configurar Visual Studio Code](#)



Para saber más

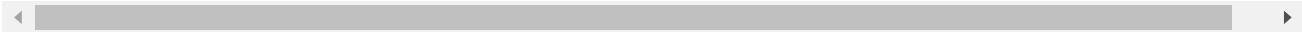
Si tienes curiosidad por saber cuáles son los plugins más usados por los profesionales puedes ver este vídeo, puedes instalar y probar los que más te llamen la atención:



00:00

16:43

Q *Plugins más usados por los profesionales* (https://www.youtube.com/watch?v=9O1PZoo0IAU&ab_channel=CarlosAzaustre-AprendeJavaScript).



5.- Integrar scripts en un HTML



Caso práctico

Antonio ha decidido ya el editor y los navegadores web que va a utilizar para programar con JavaScript, así que se pone manos a la obra y mira cuáles son los primeros pasos para integrar el nuevo código de JavaScript en el HTML.

Como Antonio ya conoce el lenguaje HTML, puesto que lo estudió en uno de los módulos que está cursando en su ciclo formativo, se centra en la parte específica de HTML que le permitirá integrar el nuevo lenguaje de programación JavaScript con el lenguaje HTML que ya conoce.

Ahora que ya conoces las herramientas que puedes utilizar para comenzar a programar en JavaScript, vamos a ver la forma de **integrar el código de JavaScript** en tu código HTML.

Los navegadores web te permiten varias opciones de inserción de código de JavaScript. Podremos insertar código usando las etiquetas `<script>` `</script>` y empleando un atributo `type` indicaremos qué tipo de lenguaje de script estamos utilizando:

Por ejemplo:

```
1 | <script type="text/javascript">
2 | // El código de JavaScript vendrá aquí.
3 | </script>
```

Esto es correcto, aunque en HTML5 ya no hace falta incluir el atributo `type` . Yo te recomiendo esta segunda forma:

```
1 | <script>
2 | // El código de JavaScript vendrá aquí.
3 | </script>
```

En estos apuntes aparecerán indistintamente con el atributo y sin el.

Otra forma de integrar el código de JavaScript es utilizar un fichero externo que contenga el código de JavaScript y referenciar dicho fichero. Ésta sería la **forma más recomendable**, ya que así se consigue una separación entre el código y la estructura de la página web y como ventajas adicionales podrás compartir código entre diferentes páginas, centralizar el código para la depuración de errores, tendrás mayor claridad en tus desarrollos, más modularidad, seguridad del código y conseguirás que las páginas carguen más rápido. La rapidez de carga de las páginas se consigue al tener el código de JavaScript en un fichero independiente, ya que si más de una página tiene que acceder a ese fichero lo cogerá automáticamente de la caché del navegador con lo que se acelerará la carga de la página.

Para ello tendremos que añadir a la etiqueta `script` el atributo `src` , con el nombre del fichero que contiene el código de JavaScript. Generalmente los ficheros que contienen código JavaScript tendrán la extensión `.js` .

Por ejemplo:

```
1 | <script type="text/javascript" src="miScript.js"></script>
```

O también:

```
1 | <script src="miScript.js"></script>
```

Si necesitas cargar más de un fichero `.js` repite la misma instrucción cambiando el nombre del fichero. Las etiquetas de `<script>` y `</script>` son obligatorias a la hora de incluir el fichero `.js`. **Atención:** no escribas ningún código de JavaScript entre esas etiquetas cuando uses el atributo `src`.

Para **referenciar el fichero origen** `.js` de JavaScript dependerá de la localización física de ese fichero. Por ejemplo en la línea anterior el fichero `miScript.js` deberá estar en el mismo directorio que el fichero `.html`. Podrás enlazar fácilmente a otros ficheros de JavaScript localizados en directorios diferentes de tu servidor o de tu dominio. Si quieres hacer una referencia absoluta al fichero, la ruta tendrá que comenzar por `http://`, en otro caso tendrás que poner la ruta relativa dónde se encuentra tu fichero `.js`.

Ejemplos:

```
1 | <script type="text/javascript" src="http://www.midominio.com/miScript.js"></script>
```

O también:

```
1 | <script src="http://www.midominio.com/miScript.js"></script>
```

En este caso estamos referenciando nuestro script mediante una referencia absoluta, que hace referencia a nuestro dominio en primer lugar y posteriormente a la ruta dentro de nuestro dominio. Es decir, estamos referenciando mediante una URL, que se refiere a nuestro dominio.

```
1 | <script type="text/javascript" src="./js/miScript.js"></script>
```

O también:

```
1 | <script src="./js/miScript.js"></script>
```

En este caso estamos referenciando nuestro script mediante una referencia relativa, que hace referencia a nuestro script desde el mismo directorio o carpeta en la que se encuentra la página `.html` en la que está incrustado dicho código, pero situado en el directorio `js`.

Cuando alguien examine el código fuente de tu página web verá el enlace a tu fichero `.js`, en lugar de ver el código de JavaScript directamente. Esto no quiere decir que tu código no sea inaccesible, ya que simplemente copiando la ruta de tu fichero `.js` y tecleándola en el navegador podremos descargar el fichero `.js` y ver todo el código de JavaScript. En otras palabras, nada de lo que tu navegador descargue para mostrar la página web podrá estar oculto de la vista de cualquier programador.

5.1.- Estructura básica de un HTML

Cuando comienzas cualquier proyecto web y no usas ningún framework, es habitual que agregues siempre las mismas líneas HTML tanto en la cabecera como en el pie del documento. En este apartado vamos a ver qué es lo mínimo que debes incluir en el archivo **index.html**.

Elementos que Debes Incluir

Aquí tienes una lista con los elementos que deberías incluir:

- **Declaración del Tipo de Documento:** Es común olvidarse de esta declaración, ya que no es imprescindible para que la página que estás creando funcione. Sin embargo, siempre está bien incluir la etiqueta `doctype` para indicar el contenido del documento a los navegadores o a los diferentes motores que vaya a consumirlo.

```
<!doctype html>
```

No se trata de una etiqueta HTML, sino que es simplemente algo que proporciona información al navegador acerca del tipo de documento que se encontrará. Esta declaración no es sensible a las mayúsculas, por lo que puedes escribirla como te plazca. Para más detalles, consulta [qué es y cómo se utiliza la etiqueta doctype](#).

- **Codificación de Caracteres:** La codificación de caracteres se indica mediante una meta etiqueta HTML. La codificación más utilizada es la **UTF-8**, que cubre el 95% de los caracteres de los idiomas existentes:

```
<meta charset="utf-8">
```

Aquí tienes más [información acerca de la codificación UTF8](#).

- **Definición del Viewport:** El **viewport** no es ningún estándar y se suele utilizar para definir el comportamiento de los navegadores en dispositivos móviles. En general, al ancho máximo del documento debe ser equivalente al ancho máximo de la pantalla del dispositivo. Para definir este valor se se usa la siguiente meta etiqueta:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Como ves, también se define la escala, que hace posible agregar un zoom si es necesario, aunque en este caso es la escala 1:1, que es la más común, ya que en general, se suelen utilizar **media queries** CSS para definir los estilos del documento.

- **Versiones Antiguas de Internet Explorer:** La meta etiqueta **X-UA-Compatible** permite definir la versión de Internet Explorer en la cual se debe renderizar la página. Esta etiqueta ya no se usa del mismo modo ni en **Internet Explorer 11** ni en el navegador **Microsoft Edge**. El navegador Internet Explorer siempre comenzará a renderizar la página usando la última versión, por lo que esta etiqueta debe situarse en la parte superior del documento. Cuando el navegador encuentre esta etiqueta, el renderizado volverá a comenzar desde cero. En general, recomiendo usar «**IE=edge**» como valor, puesto que las versiones antiguas de Internet Explorer incluyen bastantes bugs y no suelen respetar los estándares:

```
<meta http-equiv="x-ua-compatible" content="ie=edge">
```

- **Enlaces a Archivos CSS:** En general, los enlaces a los archivos **CSS** usan la etiqueta **link**, y deben colocarse en la cabecera o **head** del documento.
- **Enlaces a Archivos JavaScript:** Con respecto a lo scripts de JavaScript o archivos **JS**, existen diferentes lugares en dónde incluirlos. Podrías incluirlos directamente en la cabecera, pero si uno de los scripts modifica algún elemento HTML, seguramente salte algún error, ya que el código HTML al que hace referencia todavía no se habrá renderizado. Este es emotivo de que la librería **jQuery** incluya la famosa función `$(document).ready()`. Además, el rendimiento es bastante malo, ya que se bloqueará el análisis del código HTML. Para una explicación más detallada, consulta [por qué no debes incluir scripts en la cabecera](#). Estas son las opciones que tienes:

- ○ Incluye los archivos JavaScript en el **body**: Debes incluirlos justo antes de la etiqueta `</body>`, ya que de este modo, primero se realizará el análisis del código HTML de la página y luego se cargarán los scripts.
- ○ Incluye los archivos JavaScript en el **head**: Con la diferencia de que es imprescindible que uses los atributos `async` o `defer`, que son dos atributos que solucionan los problemas de carga y de rendimiento de los scripts en la cabecera. A día de hoy es el método más recomendable. Para más información, consulta [cómo cargar scripts usando async y defer de forma eficiente](#).

El Resultado Final

Aquí tienes la plantilla del archivo **index.html** resultante, en la que se incluyen todos los elementos de la lista:

```

1  <!DOCTYPE html>
2
3      <html lang="es">
4
5          <head>
6
7              <meta charset="utf-8" />
8
9              <meta http-equiv="x-ua-compatible" content="ie=edge" />
10
11             <meta name="viewport" content="width=device-width, initial-scale=1" />
12
13             <title>Títulos del Documento</title>
14
15             <link rel="stylesheet" href="css/archivo.css" />
16
17             <link rel="stylesheet" href="css/normalize.css" />
18
19             <link rel="icon" href="favicon.png" />
20
21         </head>
22
23         <body>
24
25             <script src="js/script.js"></script>
26
27         </body>
28
29     </html>

```



Debes conocer

Dónde incluir los scripts en un HTML

Hace un tiempo no era recomendable colocar las etiquetas `<script>` en la sección `<head>` de los documentos HTML. La mejor práctica consistía en situar las etiquetas `<script>` que incluyen código **JavaScript** justo antes la etiqueta de cierre `</body>`.

Esto es debido a que los navegadores cargan el código HTML desde la parte superior de los documentos hasta el fondo. Lo primero que se carga es la **cabecera** `<head>` y seguidamente el **body** `<body>` y todo lo que contiene en su interior. Si incluyes los scripts en la sección head, la carga del archivo JavaScript referenciado bloqueará el análisis del código HTML posterior hasta que el script se haya cargado y ejecutado. Esto puede ocasionar también problemas.

1. Si el código JavaScript de alguno de tus scripts **altera el código HTML** nada más cargarse, **no ocurrirán dichos cambios** y puede que se produzcan **errores**, ya que el código HTML **todavía no se habrá cargado**.

2. Por otro lado, el **tiempo de carga** de tu página puede que sea **considerablemente largo** es el **tamaño** de tus **archivos** JavaScript es demasiado **grande**. A mayor cantidad de código JavaScript, mayores serán los tiempos de carga. Sin embargo, si optas por incluir los archivos JavaScript justo antes de la etiqueta `</body>`, la parte más importante de la página ya se habrá renderizado cuando se carguen estos archivos.

Incluyendo los archivos JavaScript al final del documento, darás un valioso tiempo al navegador para que cargue el código HTML antes de que cargue el código JavaScript, **evitando errores** y acelerando los **tiempo de respuesta**. Además, mejorarás tu puntuación en la herramienta [Google Page Insights](#).

Sin embargo, el uso de los atributos `async` o `defer` eliminan la mayor parte de estos problemas derivados de la carga de scripts en la cabecera, ya que se descargarán de forma asíncrona. Es decir, el análisis de código HTML sucede a la vez que se descargan los scripts. Para más información acerca de estos atributos, consulta [cómo cargar scripts con async y defer](#).

Esta es la **recomendación** general:

- Si usas los atributos `async` o `defer`, incluye los scripts en el head. **Como puedes ver esta opción es más avanzada por lo que, por ahora, nos olvidaremos de ella.**
- Si no usas los atributos `async` o `defer`, incluye los scripts antes de la etiqueta de cierre `</body>`. **Por simplicidad esto es lo que haremos en las primeras unidades.**



Para saber más

Si has intentado usar `async` o `defer` seguramente te estés preguntando que por qué no ves errores cuando llevas toda la vida incluyendo los scripts en la cabecera sin usar `async` / `defer`. Muy sencillo; cuando incluyes scripts como **jQuery**, sueles usar una función que espera a que todo el código HTML se haya cargado antes de ejecutar ciertas funciones. Se trata de la función `$("#document").ready`. Aquí tienes un ejemplo:

```
1 | $("#document").ready(function(){
2 |     // Aquí va tu código
3 | });
```

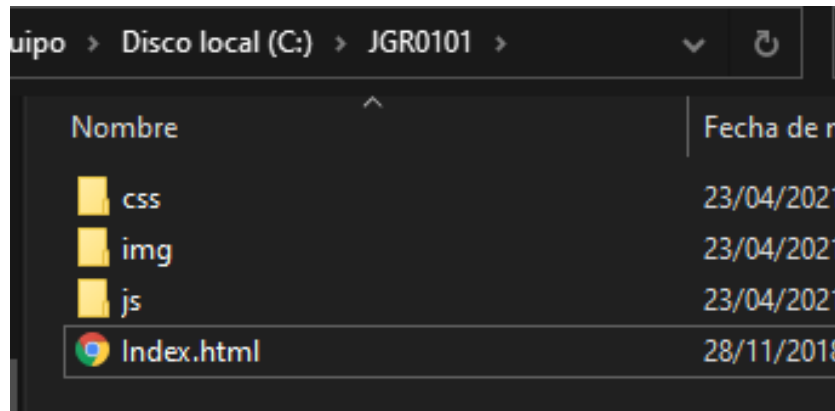
La función `ready` esperará a que el documento esté listo, tal y como su nombre indica. Muchos otros scripts incluyen funciones similares. Decir que, aunque esto en teoría funciona, afectará enormemente al rendimiento de tu página, traducándose en un menor número de visitas desde Google.

6.- Ejemplos comentados



Debes conocer

Todos los ejemplos y los ejercicios que te pida en las actividades tendrán la siguiente estructura:



Como ves, es una distribución de carpetas muy sencilla y común en los proyectos web. Separa los estilos (css), las imágenes (img) y los archivos de JavaScript (js) y llama a la página inicial index.html.

Por tanto, las etiquetas script del HTML tendrán este aspecto:

```
1 | <script src="js/primer--javascript.js"></script>
```

Para ejecutar y probar todos estos ejemplos crearás la carpeta correspondiente y copiarás el contenido del archivo index.html y de los archivos JavaScript que haya en la carpeta js.

En estos apuntes encontrarás multitud de ejemplos comentados, lo ideal es entrar en la carpeta "Contenido del directorio" y hacer clic en index.html para que se ejecute en un página externa.

[Demo: Tu primer código JavaScript.](#)

[Demo: Tu primer código JavaScript en archivo externo.](#)

[Demo: tu primera depuración de código JavaScript.](#)



Para saber más

Copia los ejemplos "Demo" y ejecútalos en local en tu navegador (se recomienda utilizar Chrome), por ahora basta con que hagas doble clic en el archivo index.html.

Los ejemplos son auto-explicativos: comprueba que hacen lo que deben hacer y cómo se ejecutan, en qué orden los hacen, si saltan instrucciones...

Te recomiendo que vayas guardando todos los proyectos de ejemplos, Demos y tareas porque podrás traerlos a los exámenes para que te sirvan de ayuda.

6.1.- Demo: Tu primer código JavaScript



Código fuente

Archivo index.html

```
1  <!DOCTYPE html>
2  <html>
3
4      <head>
5          <title>Demo: tu primer código JavaScript</title>
6          <meta name="viewport" content="width=device-width, initial-scale=1">
7
8      </head>
9
10     <body>
11         <h1>Demo: tu primer código JavaScript </h1>
12
13         <p>En este primer ejemplo puedes ver cómo incluir un script en nuestra web.</p>
14         <p>Pulsa el botón derecho de tu ratón en esta web y "Ver código fuente de esta página"
15         <p>Esta primera web muestra el mensaje "Prueba de JavaScript" en un cuadro de alerta
16         <p>Intenta analizar el orden de ejecución del código.</p>
17         <p>Observa cómo muestra el mensaje antes de "pintar" la página y cómo cuando le das a
18         <p>Pulsa F5 para recargar la página y comprobar otra vez el orden de ejecución</p>
19         <script>
20             alert("Prueba de JavaScript");
21         </script>
22         <h2>Después de JavaScript</h2>
23         <p id="demo"></p>
24
25     </body>
26 </html>
```

6.2.- Demo: Tu primer código JavaScript en archivo externo



Código fuente

Archivo index.html

```
1  <!DOCTYPE html>
2  <html>
3
4      <head>
5          <title>Demo: tu primer código JavaScript</title>
6          <meta name="viewport" content="width=device-width, initial-scale=1">
7
8      </head>
9
10     <body>
11         <h1>Demo: tu primer código JavaScript </h1>
12
13         <p>En este primer ejemplo puedes ver cómo incluir un archivo js externo en nuestra web</p>
14         <p>Pulsa el botón derecho de tu ratón en esta web y "Ver código fuente de esta página"</p>
15         <p>Pulsa encima de "primer-javascript.js" y verás tu pimer código JavaScript.</p>
16         <p>Esta primera web muestra el mensaje "Prueba de JavaScript" en un cuadro de alerta</p>
17         <p>Intenta analizar el orden de ejecución del código.</p>
18         <p>Observa cómo muestra el mensaje antes de "pintar" la página y cómo cuando le das a</p>
19         <p>Pulsa F5 para recargar la página y comprobar otra vez el orden de ejecución</p>
20
21         <script src="js/primer-javascript.js">
22
23         </script>
24
25         <h2>Después de JavaScript</h2>
26
27         <p id="demo"></p>
28
29     </body>
30 </html>
```

Archivo primer-javascript.js

```
1 | alert("Prueba de JavaScript");
```

7.- Depurar código JavaScript con Chrome

Este tutorial te enseña el flujo de trabajo básico para depurar cualquier problema de JavaScript en DevTools.

Sigue leyendo o mira la versión en video de este tutorial a continuación.

Debugging JavaScript - Chrome DevTools 101



Depurar es fundamental en cualquier lenguaje, pero en JavaScript lo es aún más porque usa tipos de datos dinámicos y es complicado saber qué es cada variable en cada momento.

No se puede programar en JavaScript sin saber depurar correctamente. Por eso, esta lección es una de las más importantes del módulo.

Paso 1: reproduce el error

Encontrar una serie de acciones que reproduzcan consistentemente un error es siempre el primer paso para la depuración.

1.

Copia el código Demo: tu primera depuración de código JavaScript. Ábrelo en una nueva pestaña en una nueva pestaña.

[7.1.- Demo: tu primera depuración de código JavaScript.](#)

2.

Ingresa `5` en el cuadro de texto **Número 1**.

3.

Ingresa `1` en el cuadro de texto **Número 2**.

4.

Haz clic en **Agregar número 1 y número 2**. La etiqueta debajo del botón dice `5 + 1 = 51`. El resultado debería ser `6`. Este es el error que vas a solucionar.

Figura 1 . El resultado de $5 + 1$ es 51. Debería ser 6.

Paso 2: familiarízate con la interfaz de usuario del panel de fuentes

DevTools proporciona muchas herramientas diferentes para diferentes tareas, como cambiar CSS, perfilar el rendimiento de carga de la página y monitorear las solicitudes de red. El panel **Fuentes** es donde depura JavaScript.

1.

Abre DevTools presionando Comando + Opción + I (Mac) o Control + Shift + I (Windows, Linux). También puedes pulsar F12. Este acceso directo abre el panel de la **consola** .

Figura 2 . El panel de la **consola**

2.

Haga clic en la pestaña **Fuentes** .

Figura 3 . El panel de **fuentes**

La interfaz de usuario del panel de **Fuentes** tiene 3 partes:

Figura 4 . Las 3 partes de la interfaz de usuario del panel de **Fuentes**

1. El panel del **navegador de archivos** . Todos los archivos que solicita la página se enumeran aquí.
2. El panel del **editor de código** . Después de seleccionar un archivo en el panel **Navegador de archivos** , el contenido de ese archivo se muestra aquí.
3. El panel **Depuración de JavaScript** . Varias herramientas para inspeccionar el JavaScript de la página. Si la ventana de DevTools es amplia, este panel se muestra a la derecha del panel **Editor de código** .

Paso 3: pausa el código con un punto de interrupción

Un método común para depurar un problema como este es insertar muchas `console.log()` declaraciones en el código para inspeccionar los valores a medida que se ejecuta el script. Por ejemplo:

```
1 function updateLabel() {  
2     var addend1 = getNumber1();  
3     console.log('addend1:', addend1);  
4     var addend2 = getNumber2();  
5     console.log('addend2:', addend2);  
6     var sum = addend1 + addend2;  
7     console.log('sum:', sum);  
8     label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;  
9 }
```

El `console.log()` método puede hacer el trabajo, pero los **puntos de interrupción** pueden hacerlo más rápido. Un punto de interrupción te permite pausar tu código en medio de su ejecución y examinar todos los valores en ese momento. Los puntos de interrupción tienen algunas ventajas sobre el `console.log()` método:

- Con `console.log()` , debe abrir manualmente el código fuente, encontrar el código relevante, insertar las `console.log()` declaraciones y luego volver a cargar la página para ver los mensajes en la Consola. Con los puntos de interrupción, puedes hacer una pausa en el código relevante sin siquiera saber cómo está estructurado el código.
- En tus `console.log()` declaraciones, debes especificar explícitamente cada valor que deseas inspeccionar. Con los puntos de interrupción, DevTools te muestra los valores de todas las variables en ese momento. A veces, hay variables que afectan tu código y que ni siquiera conoces.

En resumen, los puntos de interrupción pueden ayudarte a encontrar y corregir errores más rápido que el `console.log()` método.

Si das un paso atrás y piensas en cómo funciona la aplicación, puedes hacer una suposición fundamentada de que la suma incorrecta (`5 + 1 = 51`) se calcula en el `click` detector de eventos que está asociado al botón **Agregar número 1 y Número 2** . Por lo tanto, probablemente desees pausar el código en el momento en que se `click` ejecuta el oyente. **Los puntos de interrupción de Event Listener** te permiten hacer exactamente eso:

1.

En el panel **Depuración de JavaScript** , haga clic en **Event Listener Breakpoints** para expandir la sección. DevTools revela una lista de categorías de eventos expandibles, como **Animación** y **Portapapeles** .

2.

Junto a la categoría de eventos **Mouse** , haga clic en **Expandir** . DevTools revela una lista de eventos del mouse, como **hacer clic** y **mousedown** . Cada evento tiene una casilla de verificación junto a él.

3.

Marca la casilla de verificación de **clic** . DevTools ahora está configurado para pausar automáticamente cuando se ejecuta *cualquier* `click` detector de eventos.

Figura 5 . La casilla de verificación de **click** está habilitada

4.

De vuelta en la demostración, haz clic en **Agregar el número 1 y el número 2** nuevamente. DevTools pausa la demostración y resalta una línea de código en el panel **Fuentes** . DevTools debe pausarse en esta línea de código:

```
function onClick() {
```

Si está pausado en una línea diferente de código, presione **Reanudar ejecución de script** hasta que esté pausado en la línea correcta.

Nota : Si hiciste una pausa en una línea diferente, tienes una extensión de navegador que registra un `click` detector de eventos en cada página que visita. Te detuviste en el `click` oyente de la extensión . Si usas el modo incógnito para navegar en privado , lo que deshabilita todas las extensiones, puede ver que se detiene en la línea de código correcta cada vez.

Los puntos de interrupción de **Event Listener** son solo uno de los muchos tipos de puntos de interrupción disponibles en DevTools. Vale la pena memorizar todos los tipos diferentes, porque cada tipo finalmente te ayuda a depurar diferentes escenarios lo más rápido posible.

Paso 4: recorre el código

Una causa común de errores es cuando un script se ejecuta en el orden incorrecto. Recorrer tu código te permite recorrer la ejecución de su código, una línea a la vez, y averiguar exactamente dónde se está ejecutando en un orden diferente al que esperaba. Pruébalo ahora:

1.

En el panel **Fuentes** de DevTools, haz clic en **Pasar a la siguiente llamada de función** para recorrer la ejecución de la `onClick()` función, una línea a la vez. DevTools destaca la siguiente línea de código:

```
if (inputsAreEmpty()) {
```

2.

Haz clic en **Pasar a la siguiente llamada de función** . DevTools se ejecuta `inputsAreEmpty()` sin entrar en él. Observe cómo DevTools omite algunas líneas de código. Esto se debe a que se `inputsAreEmpty()` evaluó como falso, por lo `if` que el bloque de código de la declaración no se ejecutó.

Esa es la idea básica de recorrer el código. Si observas el código `get-started.js` , puedes ver que el error probablemente esté en algún lugar de la `updateLabel()` función. En lugar de recorrer cada línea de código, puede usar otro tipo de punto de interrupción para pausar el código más cerca de la ubicación probable del error.

Paso 5: establece un punto de interrupción de línea de código

Los puntos de interrupción de línea de código son el tipo de punto de interrupción más común. Cuando tengas una línea de código específica en la que desees hacer una pausa, usa un punto de interrupción de línea de código:

1.

Mira la última línea de código en `updateLabel()` :

```
label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;
```

2.

A la izquierda del código, puede ver el número de línea de esta línea de código en particular, que es **32** . Haga clic en **32** . DevTools pone un ícono azul encima de **32** . Esto significa que hay un punto de interrupción de línea de código en esta línea. DevTools ahora siempre hace una pausa antes de que se ejecute esta línea de código.

3.

Haz clic en **Reanudar la ejecución del script** . La secuencia de comandos continúa ejecutándose hasta que llegue a la línea 32. En las líneas 29, 30 y 31, grabados DevTools fuera de los valores `addend1` , `addend2` y `sum` a la derecha del punto y coma de cada línea.

Figura 6 . DevTools se detiene en el punto de interrupción de la línea de código en la línea 32

Paso 6: verifica los valores de las variables

Los valores de `addend1` , `addend2` y `sum` parecen sospechosos. Están entre comillas, lo que significa que son cadenas. Esta es una buena hipótesis para explicar la causa del error. Ahora es el momento de recopilar más información. DevTools proporciona muchas herramientas para examinar los valores de las variables.

Método 1: el panel de alcance

Cuando está en pausa en una línea de código, el panel **Ámbito** le muestra qué variables locales y globales están definidas actualmente, junto con el valor de cada variable. También muestra las variables de cierre, cuando corresponde. Haga doble clic en un valor de variable para editarlo. Cuando no está en pausa en una línea de código, el panel **Ámbito** está vacío.

Figura 7 . El panel de **alcance**

Método 2: Ver expresiones

La pestaña **Ver expresiones** le permite monitorear los valores de las variables a lo largo del tiempo. Como su nombre lo indica, las expresiones de observación no se limitan solo a las variables. Puede almacenar cualquier expresión de JavaScript válida en una expresión de observación. Prueballo ahora:

1.
Haz clic en la pestaña **Ver** .
2.
Haz clic en **Agregar expresión** .
3.
Escribe `typeof sum` .
- 4.

Presiona Entrar. Muestra DevTools `typeof sum: "string"` . El valor a la derecha de los dos puntos es el resultado de su expresión de reloj.

Figura 8 . El panel Ver expresión (abajo a la derecha), después de crear la `typeof sum` expresión Ver. Si la ventana de DevTools es grande, el panel Ver expresión está a la derecha, encima del panel **Puntos de interrupción del escucha de eventos** .

Como se sospecha, `sum` se evalúa como una cadena, cuando debería ser un número. Ahora ha confirmado que esta es la causa del error.

Método 3: la consola

Además de ver `console.log()` mensajes, también puedes utilizar la consola para evaluar declaraciones de JavaScript arbitrarias. En términos de depuración, puede usar la Consola para probar posibles correcciones de errores. Pruébalo ahora:

1.

Si no tienes el cajón de la consola abierto, presione Escape para abrirlo. Se abre en la parte inferior de la ventana de DevTools.

2.

En la consola, escribe `parseInt(addend1) + parseInt(addend2)` . Esta declaración funciona porque está en pausa en una línea de código donde `addend1` y `addend2` está dentro del alcance.

3.

Presiona Entrar. DevTools evalúa la declaración y la imprime `6` , que es el resultado que espera que produzca la demostración.

Figura 9 . El cajón de la consola, después de evaluar `parseInt(addend1) + parseInt(addend2)` .

Paso 7: aplica una corrección

Has encontrado una solución para el error. Todo lo que queda es probar su solución editando el código y volviendo a ejecutar la demostración. No es necesario que salga de DevTools para aplicar la solución. Puede editar el código JavaScript directamente dentro de la interfaz de usuario de DevTools. Pruebalo ahora:

1. Haz clic en **Reanudar la ejecución del script** .
2. En el **Editor de código** , reemplace la línea 31,, `var sum = addend1 + addend2` con `var sum = parseInt(addend1) + parseInt(addend2)` .
3. Presiona Comando + S (Mac) o Control + S (Windows, Linux) para guardar su cambio.
4. Haz clic en **Desactivar puntos de interrupción** . Cambia de azul para indicar que está activo. Mientras esto está configurado, DevTools ignora los puntos de interrupción que haya establecido.
5. Prueba la demostración con diferentes valores. La demostración ahora calcula correctamente.

Precaución

Precaución: este flujo de trabajo solo aplica una corrección al código que se ejecuta en su navegador. **No arreglará el código para todos los usuarios que visiten su página. Para hacer eso, necesitas corregir el código que está en sus servidores.**

Pasos siguientes

¡Felicidades! Ahora sabes cómo aprovechar al máximo las DevTools de Chrome al depurar JavaScript. Las herramientas y métodos que aprendió en este tutorial pueden ahorrarle innumerables horas.

Este tutorial solo le mostró dos formas de establecer puntos de interrupción. DevTools ofrece muchas otras formas, que incluyen:

- Puntos de interrupción condicionales que solo se activan cuando la condición que proporciona es verdadera.
- Puntos de interrupción en excepciones detectadas o no detectadas.
- Puntos de interrupción de XHR que se activan cuando la URL solicitada coincide con una subcadena que proporcionas.

Ampliación

Si te has quedado con ganas de más puedes seguir con: [Pausar su código con puntos de interrupción](#) para saber cuándo y cómo usar cada tipo.

7.1.- Demo: tu primera depuración de código JavaScript.



Código fuente

Archivo index.html

```
1  <!doctype html>
2  <html>
3
4    <head>
5      <title>Demo: tu primera depuración de código JavaScript</title>
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7
8      <style>
9
10         h1 {
11             font-size: 1.5em
12         }
13
14         input, button {
15             min-width: 72px;
16             min-height: 36px;
17             border: 1px solid grey;
18         }
19
20         label, input, button {
21             display: block;
22         }
23
24         input {
25             margin-bottom: 1em;
26         }
27
28     </style>
29
30 </head>
31
32 <body>
33
34     <h1>Demo: tu primera depuración de código JavaScript</h1>
35
36     <p>En este primer ejemplo puedes ver cómo incluir un archivo js externo en nuestra web.</p>
37     <p>Pulsa el botón derecho de tu ratón en esta web y "Ver código fuente de esta página", a
38     <p>Pulsa encima de "primer-javascript.js" y verás tu pimer código JavaScript.</p>
39     <p>Esta primera web pretende sumar dos números. Pero si la pruebas un poco verás que, en
40     <p>Intenta analizar cómo comprueba que se ha pulsado el botón Suma, cómo recoge el primer
41     <p>No te asustes, no hace falta que comprendas aún este código solamente que te vayas fan
42     <p>Intenta descargar el archivo "index.html", junto con el archivo "primer-javascript.js"
43
44     <p>En el siguiente apartado aprenderás a depurar este código, para comprender cómo arregl
45
46     <label for="num1">Number 1</label>
47
48     <input placeholder="Number 1" id="num1">
```

```
<label for="num2">Number 2</label>

<input placeholder="Number 2" id="num2">

<button>Suma Number 1 and Number 2</button>

<h2></h2>

<script src="js/primer-debug-javascript.js"></script>

</body>

</html>
```

Archivo primer-debug-javascript.js

```
1  /* Copyright 2016 Google Inc.
2
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7
8  *
9  * http://www.apache.org/licenses/LICENSE-2.0
10
11  *
12  * Unless required by applicable law or agreed to in writing, software
13  * distributed under the License is distributed on an "AS IS" BASIS,
14  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15  * See the License for the specific language governing permissions and
16  * limitations under the License. */
17
18 function onClick() {
19
20   if (inputsAreEmpty()) {
21     label.textContent = 'Error: one or both inputs are empty.';
22     return;
23   }
24   updateLabel();
25 }
26
27 function inputsAreEmpty() {
28
29   if (getNumber1() === '' || getNumber2() === '') {
30     return true;
31   } else {
32     return false;
33   }
34 }
35
36 function updateLabel() {
37
38   var addend1 = getNumber1();
39   var addend2 = getNumber2();
```

```
    var sum = addend1 + addend2;

    label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;
}

function getNumber1() {
    return inputs[0].value;
}

function getNumber2() {
    return inputs[1].value;
}

var inputs = document.querySelectorAll('input');
var label = document.querySelector('h2');
var button = document.querySelector('button');
button.addEventListener('click', onClick);
```

7.2.- Para saber más: integrar toda la depuración en Visual Studio Code



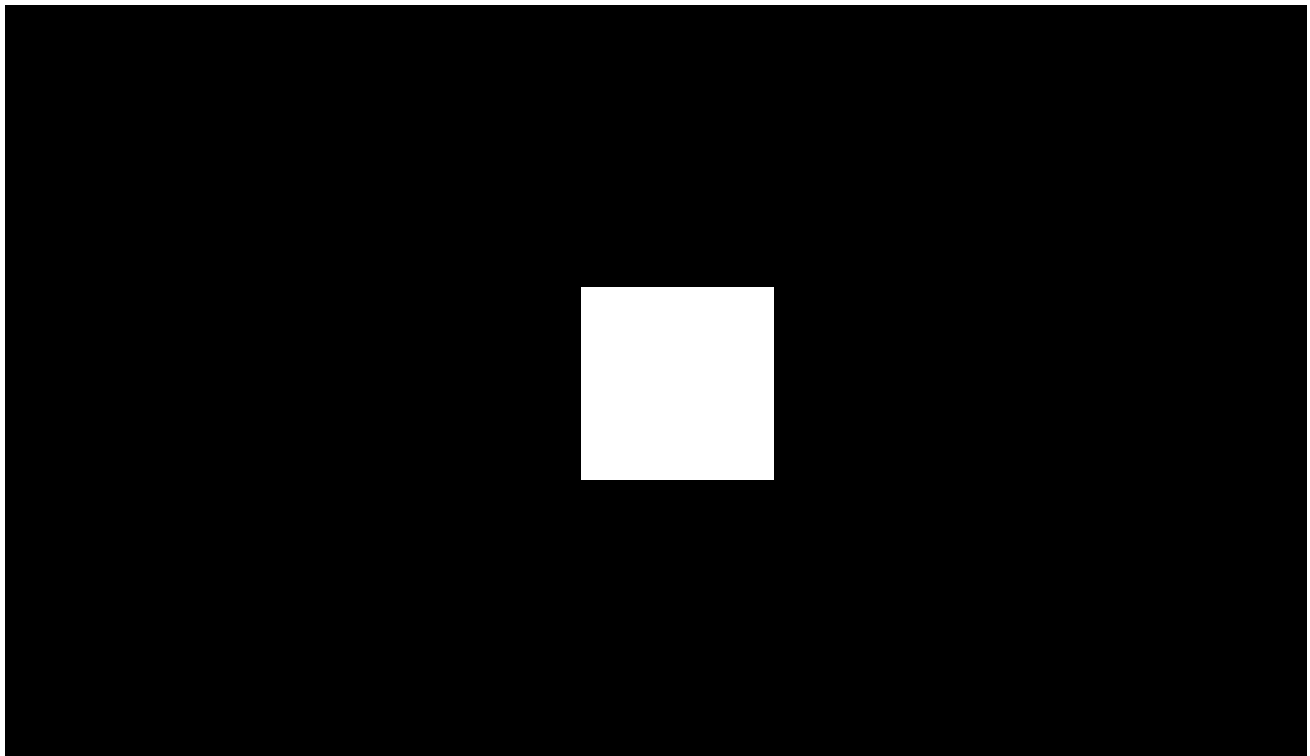
Para saber más

Hasta hace poco, yo mismo he programado mis proyectos en VS Code y los he depurado desde Chrome, tal y como te acabo de enseñar. No es un mal modo de organizar el flujo de trabajo y tiene gran la ventaja de que estamos depurando en el mismo navegador que va a ejecutar el código en los usuarios finales.

Sin embargo, tal y como habrás comprobado resulta un poco tedioso escribir el código en VS Code y tener que poner los puntos de interrupción y consultar la consola en el navegador. ¿Existe alguna manera de poder integrar todo en VS Code? Hasta hace poco no, pero en la última versión han incorporado esta esperada función.

Como todos los apartados "Para saber más" no se te pedirá nada de esto en ningún ejercicio ni examen, sigue éste tutorial solo si has comprendido y practicado ya la depuración en Chrome y si tienes curiosidad por conseguir un flujo de trabajo más cómodo.

El proceso para conseguirlo está explicado en este vídeo:





00:00

05:10

🔍 [Learn how to debug web projects end-to-end in VS Code in 5 minutes \(https://www.youtube.com/watch?v=vRNdnv_-X18\)](https://www.youtube.com/watch?v=vRNdnv_-X18)

Si te has perdido en alguno de los pasos, te los pongo en texto en Español:

- Abre un proyecto dentro de VS Code abriendo su carpeta. El código Demo utilizado es una  aplicación de lista de tareas que puedes [\(listaDeTareas.zip\)](#)  [descargar aquí](#).
([listaDeTareas.zip](#)).

- Abre una Terminal dentro de VS Code e inicia un servidor local. Puedes usar algún plugin de VScode o crearlo con XAMP.
- En lugar de abrir el proyecto en el navegador, tal y como hemos aprendido, utiliza el flujo (<https://code.visualstudio.com/docs/nodejs/browser-debugging>) de trabajo (<https://code.visualstudio.com/docs/nodejs/browser-debugging>) de ejecución y depuración (<https://code.visualstudio.com/docs/nodejs/browser-debugging>) de VS Code para abrir una instancia de navegador dedicada para depurar este proyecto. Si tengo un navegador basado en Chromium en mi dispositivo y lo tengo configurado como mi navegador predeterminado, este será el que use VS Code. En el caso del vídeo, esto abre una nueva ventana de Microsoft Edge, ya que este es su navegador principal. El vídeo utiliza Edge para mostrar una funcionalidad adicional más adelante que solo es compatible con este navegador basado en Chromium.
- Tener una instancia de navegador dedicada también significa que ahora obtienes la funcionalidad de la Consola de herramientas de desarrollo (<https://docs.microsoft.com/microsoft-edge/devtools-guide-chromium/console/>) del navegador (<https://docs.microsoft.com/microsoft-edge/devtools-guide-chromium/console/>) dentro de la Consola de depuración de Visual Studio Code. La consola de depuración muestra cualquier mensaje de console.log (<https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide-chromium/console/console-log>) de tu JavaScript y también puedes interactuar con el documento en el navegador usando los métodos de conveniencia de (<https://docs.microsoft.com/microsoft-edge/devtools-guide-chromium/console/utilities>) la consola (<https://docs.microsoft.com/microsoft-edge/devtools-guide-chromium/console/utilities>) como `$` para `querySelector()`. Tengo acceso completo al objeto de la ventana y puedo cambiar el DOM y los estilos del documento de forma programática.
- Además de la depuración de `console.log()`, también tienes la oportunidad de usar la depuración de puntos de interrupción (<https://code.visualstudio.com/Docs/editor/debugging>), lo que me da mucha más información y también tiene la ventaja de detener la ejecución de mi script hasta que descubra qué está pasando.
- Como está usando Microsoft Edge como navegador de depuración, también obtiene un botón de inspección en la barra de herramientas del depurador. Este me da acceso a las herramientas de desarrollo de Edge directamente dentro de Visual Studio Code. Esta funcionalidad está impulsada por la extensión Edge Tools para VS Code, (<https://aka.ms/devtools-for-code>) que se instala la primera vez que selecciono el botón inspeccionar.
- Una vez que se ejecuta la extensión, puedes acceder y cambiar el DOM y el CSS del proyecto actual directamente dentro de Visual Studio Code usando las mismas herramientas que normalmente usaríamos en el navegador.
- También tienes acceso a la herramienta de red para inspeccionar cualquier solicitud de red y ver si hay algún problema de contenido que no se carga.
- Para automatizar el proceso de iniciar el navegador y abrir la dirección de localhost la próxima vez, también puedes obtener la extensión del depurador para generar un archivo `launch.json` para mí. Una vez que lo tenga, la próxima vez que todo lo anterior ocurra automáticamente.

Si deseas obtener más información sobre el depurador de JavaScript en VS Code, puede consultar la documentación aquí o hablar con [Connor Peet](#) en Twitter. Para la integración de Edge DevTools para VS Code, puede [leer la documentación de la extensión](#) y [verificar el código en GitHub](#).

Un agradecimiento especial al equipo involucrado en todo este trabajo, James Lissiak, Michael Liao, Vidal Guillermo Diazleal Ortega, Brandon Goddard, Olivia Flynn, Tony Ross, Rob Paveza, Jason Stephen, Connor Peet y por supuesto el equipo de VS Code.

[Este tutorial es una traducción de un artículo original de Chris Heilmann](#)

Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
(<http://creativecommons.org/licenses/by-sa/4.0/>).