TEMA

Tema 1. Bases para el desarrollo web en entorno cliente

Desarrollo de aplicaciones web

Desarrollo web en entorno cliente

Autor: Cristian Catalán

Linkia FP

Tema 1: Bases para el desarrollo web en entorno cliente

¿Qué aprenderás?

- Identificar la arquitectura cliente servidor y sus principales tecnologías.
- Utilizar herramientas para la programación web.
- Integrar JavaScript en un documento HTML.
- Programar en JavaScript utilizando los principales controles deflujo.
- Programar operaciones básicas de entrada y salida de datos con JavaScript.

¿Sabías que…?

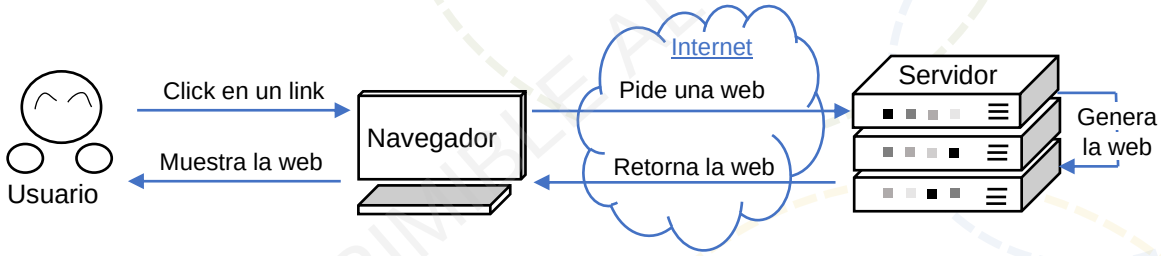
- Netscape Navigator fue uno de los navegadores más importantes en los inicios de la WWW.
- Java y JavaScript solo comparten similitudes en el nombre
- NodeJs permite ejecutar código JavaScript como una aplicación de escritorio sin un navegador web.

1.1. Paradigma de la programación en entorno cliente

Cuando hablamos de programación en entorno cliente nos tenemos que situar en la WWW. En este contexto entendemos como la entidad "cliente" a cualquier programa que se conecte a internet para obtener algún tipo de información.

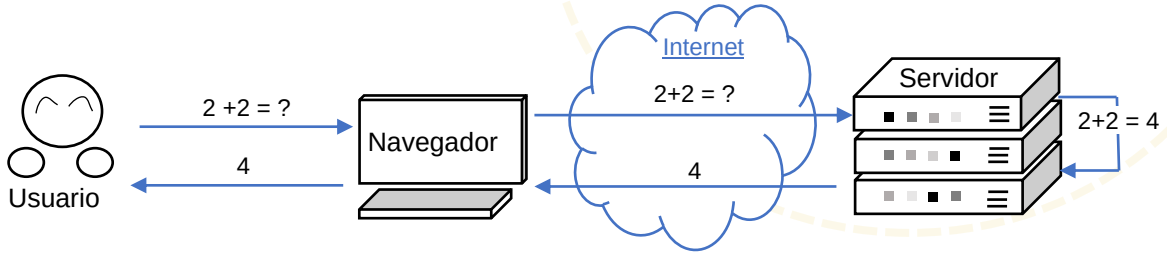
Cada vez que un "cliente" se conecta a internet para pedir información, en realidad está pidiendo un "recurso" a una dirección de internet. Por ejemplo, si el navegador pide a internet la página web https://linkiafp.es/index.php, de forma simplificada podemos pensar que está pidiendo el recurso "index.php" al ordenador ubicado en la dirección http://linkiafp.es. Esta entidad encargada de servir un recurso pedido es el "servidor".

En los inicios de la WWW los navegadores se limitaban a mostrar una interfaz amigable para que el usuario pudiera, a través de internet, acceder a contenidos almacenados en servidores y mostrarlos. Toda la programación se realizaba en los servidores, quienes generaban los recursos pedidos (páginas web, imágenes, vídeos, etc..) y lo retornaban al navegador (el cliente).



Ejemplo de comunicación entre cliente (Navegador) y el servidor

Siguiendo con éste paradigma, si quisiéramos montar una calculadora web, el navegador solo se encargaría de mostrar la interfaz a través de la cual el usuario podría escribir los valores y operaciones a realizar. Cada vez que el usuario quisiera realizar una operación se enviaría una petición al servidor pidiendo el resultado de la operación. Finalmente servidor sería el encargado de operar, generar y retornar la respuesta hacia el navegador.



Si el navegador no puede ejecutar código, todas las operaciones la realiza el servidor

Si tenemos presente que realizar peticiones al servidor es una operación muy lenta y costosa, no es de extrañar que fueran apareciendo distintos lenguajes de programación para ejecutar ciertas operaciones en el mismo navegador, obteniendo un resultado casi inmediato y mejorando la experiencia del usuario con la web.

A continuación profundizaremos en el paradigma cliente-servidor comprendiendo su arquitectura, conociendo algunos de los principales lenguajes y tecnologías que intervienen, viendo sus capacidades y las herramientas que utilizaremos para su programación.

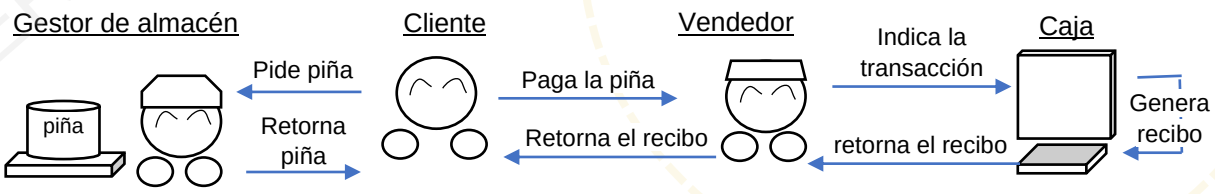
1.1.1. Arquitectura de N capas

Como hemos visto en la WWW intervienen múltiples entidades con distintas y variantes tecnologías cada vez que queremos pedir un recurso. Así que tuvieron que buscar un paradigma que limitara qué entidades podrían interactuar entre ellas y un protocolo común para su comunicación.

La arquitectura de N capas fue el paradigma que indicó qué entidades podrían interactuar entre ellas. En una arquitectura de N capas cada capa puede tener distintas entidades con distintas tecnologías, pero todas las entidades que formen una capa han de tener una misma funcionalidad claramente definida. Además una capa solo puede interactuar con su capa anterior y su capa posterior utilizando unos protocolos de comunicación claramente definidos.

Una arquitectura de N capas puede ser utilizada también en la vida real, por ejemplo la podemos utilizar para gestionar la compra de un producto en un colmado. En el siguiente ejemplo vemos una arquitectura formada por 4 capas: gestor de almacén, cliente, vendedor y caja. Como hemos visto cada capa solo puede interactuar con su capa anterior y posterior, así que:

- El Gestor de almacén solo puede interactuar con el cliente
- El cliente solo puede interactuar con el gestor de almacén y el vendedor
- El vendedor solo puede interactuar con el cliente y la Caja
- La Caja solo puede interactuar con el Vendedor



Ejemplo de una arquitectura de 4 capas en un colmado.

Una vez especificados los protocolos de comunicación entre las capas, la tecnología que utilice cada entidad que forme una capa es irrelevante. Es decir, los gestores de almacén pueden ser humanos, robots, marcianos o chimpancés siempre y cuando entiendan los protocolos de comunicación establecidos para interactuar con los clientes. Y lo mismo ocurre para cada una del resto de capas.

1.1.2. Arquitectura cliente - servidor

En la WWW se utiliza una arquitectura de como mínimo 2 capas para servir los recursos pedidos por internet. Esta arquitectura se llama de "cliente-servidor" porque las dos capas que la forman son la capa de "cliente" y la capa del "servidor". En la capa de "cliente" encontraremos todos aquellos programas que necesiten pedir algún tipo de recurso y en la capa de "servidor" se situarán todos aquellos programas capaces de dar algún tipo de servicio.

En la capa *cliente* podemos encontrar comúnmente:

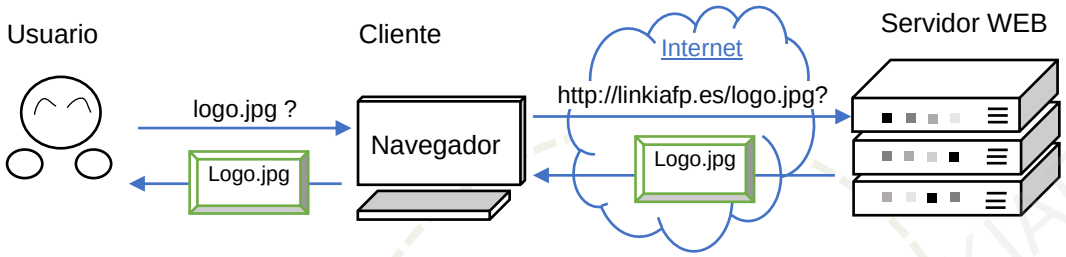
- Navegadores web para navegar por la WWW
- Clientes FTP para transferir ficheros
- Clientes de chat para chatear por internet
- Clientes de correo para gestionar emails.
- ..

En la capa de *servidor* encontraremos servidores capaces de ofrecer los recursos pedidos por los *clientes*:

- Servidores web, capaces de servir una gran variedad de tipos de archivos (html, pdf, jpg, mpeg, etc..)
- Servidores FTP especializados en servir y gestionar archivos.
- Servidores de Chat
- Servidores de correo electrónico especializados en servir y gestionar emails.

Es muy común que se añada a ésta arquitectura alguna capa antes del *cliente* o después del *servidor*. Por ejemplo: si un usuario quiere obtener una imagen, el usuario se representa como una nueva capa añadida delante del *cliente* como una capa que va a interaccionar solamente con el "cliente". De ésta forma el "cliente" pedirá la imagen al "servidor" y el "servidor" la va a retornar al "cliente"

En el siguiente diagrama se muestra a modo de ejemplo la arquitectura cliente-servidor utilizada para pedir una imagen a partir de un navegador web:



Ejemplo de una arquitectura simple cliente-servidor.

A través de internet y en la mayoría de ocasiones cualquier recurso ofrecido por un servidor tiene asociada una dirección única en formato URL (Localizador Uniforme de Recursos). Aunque existen otros tipos identificadores, la URL es la forma más común para identificar un recurso. Cualquier URL sigue la siguiente sintaxis:

<protocolo>://<nombre>:<password>@<servidor>:<puerto>/<direccion>?<parámetro1>=<valor1>&<parámetro2>=<valor2> … &<parámetroN>=<valorN>

Por ejemplo:

ftp://yo:miPasword@ftp.micasa.com/ficheros/fotos_boda.zip

https://linkiafp.es/contacto/

http://en.wikipedia.org/wiki/URL

https://www.google.es?#hl=es&q=ola+k+ase

mailto:micorreo@linkiafp.es

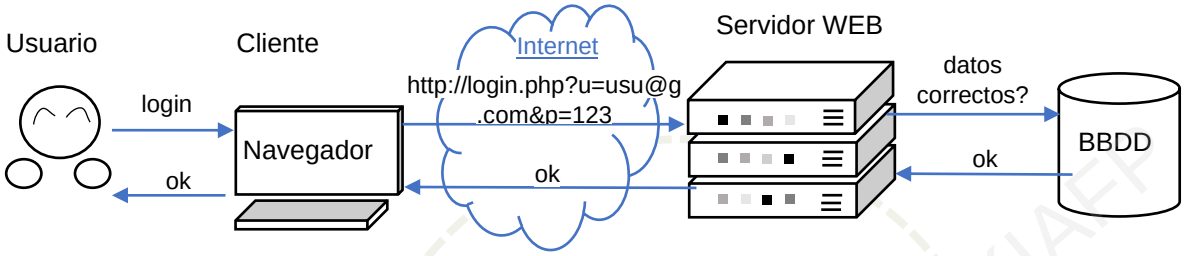
file:///c:/WINDOWS/mi_archivo.txt

El protocolo indicará el tipo de servicio que se quiere obtener. Como se puede observar no todos los campos son necesarios, pues en muchas ocasiones son irrelevantes o son añadidos por el mismo cliente o incluso el servidor.

Por ejemplo, actualmente en la mayoría de navegadores si introducimos una dirección web sin indicar el protocolo “https://”, será el mismo navegador quien lo añada por nosotros.

Siempre que el *servidor* necesite acceder a una base de datos para obtener un recurso, o comprobar si el usuario tiene las credenciales necesarias, añadiremos entonces al diagrama anterior una nueva capa detrás del servidor web.

En el siguiente diagrama se muestra a modo de ejemplo la arquitectura cliente-servidor utilizada para validar el *login* de un usuario comprobando que los datos de *usuario* y password introducidos por un usuario son los mismos que los almacenados en la base de *datos*.



Ejemplo de una arquitectura cliente-servidor con acceso a una base de datos

Como podemos observar en el gráfico anterior, el *cliente* nunca va a poder tener acceso a la base de datos con la que trabaja el servidor, es decir, que siempre que el navegador necesite acceder a la base de datos va a enviar la petición al *servidor Web* quien se encargará de conectarse con la base de datos y retornar la respuesta al navegador.

1.1.3. Pidiendo una página web al servidor

Cuando el navegador pide una página web al servidor responde solamente con el HTML de la página. Cuando el navegador obtiene el HTML empieza a interpretar su código y cada vez que se encuentra con un recurso adicional incrustado en el HTML (archivos JavaScript, CSS, imágenes, videos, etc..) realiza una nueva petición al servidor para que el servidor le retorne ese recurso. Cada vez que el navegador recibe un recurso incrustado lo interpreta según el tipo de recurso hasta que la página web se termina de formar.

En el siguiente diagrama de comunicación se muestra se muestran las distintas peticiones que ha de realizar un navegador web para cargar por completo un documento HTML formado por un documento JS y dos imágenes:

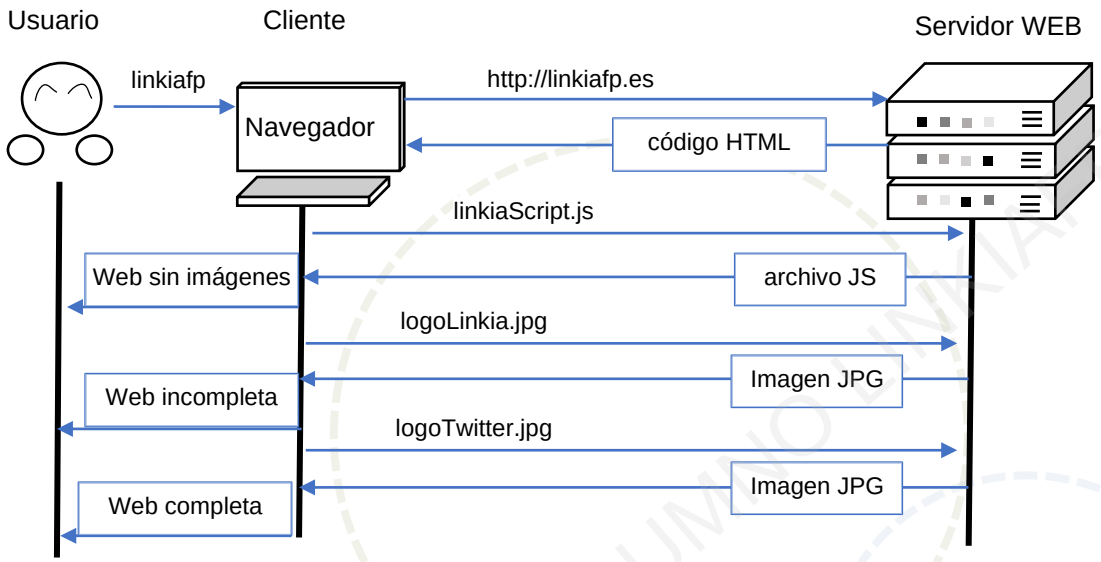


Diagrama de comunicación durante la carga de una web formada por un JS y dos imágenes.

Estas peticiones de recursos incrustados, el navegador las puede realizar de forma síncrona (hasta que no recibe la respuesta no sigue creando la web) o asíncrona dependiendo del tipo de recurso y el navegador (para nuestros ejercicios supondremos que se realiza de forma síncrona).

Si el navegador pide una página web que contiene código PHP, lo primero que deberá hacer el servidor es ejecutar ese PHP, y posteriormente retornar el HTML resultante. Cuando el navegador reciba el HTML resultante sabrá si ha de pedir más recursos o no al servidor.

En el siguiente diagrama de comunicación se muestra un ejemplo de una web con código PHP que comprueba si el usuario logeado es correcto. En este caso supondremos que es correcto y mostrará una imagen de correcto.

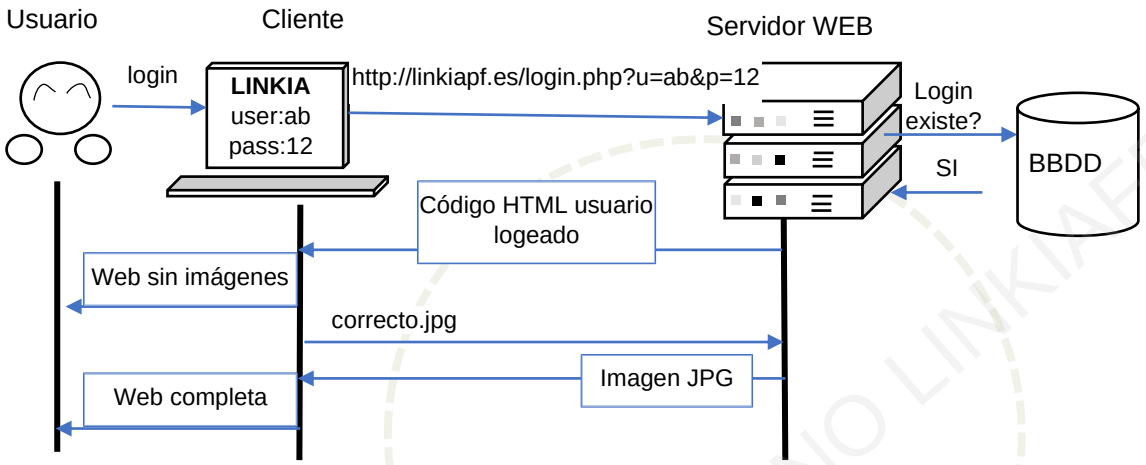
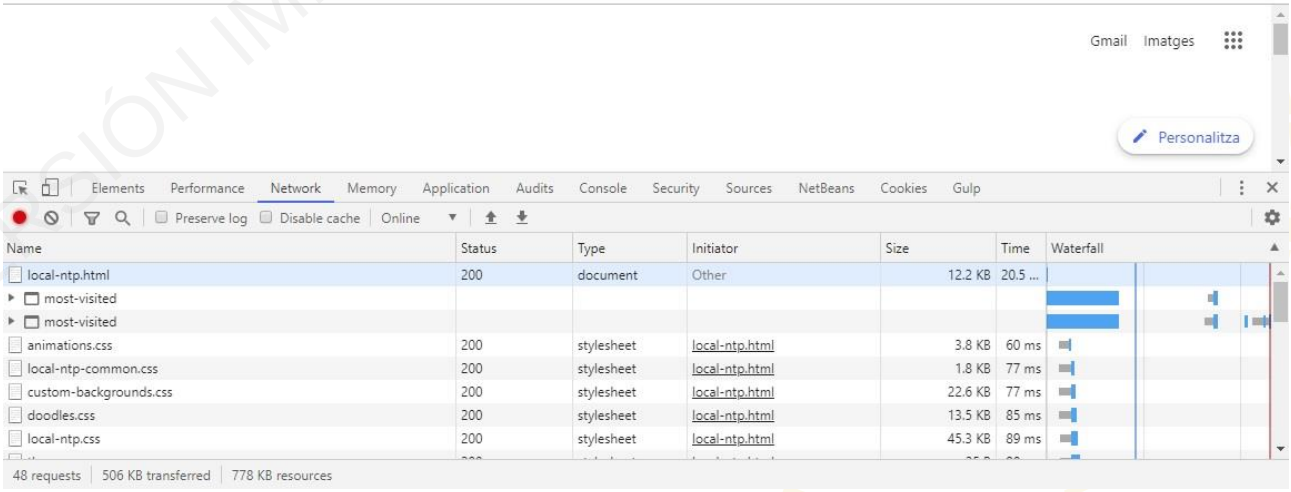


Diagrama de comunicación durante la carga de una web generada por PHP con una imagen.

Las herramientas de desarrollador que incorporan los navegadores suelen contener inspectores de red que permiten controlar todas las peticiones que se realizan en un sitio web. De cada petición podremos ver el tiempo que ha tardado, y si clicamos en ella se nos mostrará más información como las cabeceras de la petición y los datos que nos ha respondido.

En la siguiente captura se muestra (utilizando el inspector de red de Chrome) todas las peticiones que ha tenido que hacer el navegador para cargar al web de Google.



Peticiones al servidor vistas con el inspector de red de Chrome

1.1.4. Lenguajes y tecnologías

Como hemos visto, en una arquitectura cliente-servidor pueden coexistir distintas tecnologías en cada una de las capas que la forman. A continuación veremos las tecnologías más relevantes para nosotros utilizadas en el desarrollo web:

1.1.4.1. Tecnologías en la capa cliente

Como hemos visto en la capa de cliente es donde se ejecutará el navegador web. Sin entrar en mucho detalle algunos de los principales navegadores que podemos encontrar hoy en día son:

- **Google Chrome**: posiblemente el más utilizado hoy en día. Desarrollado por Google se basa en su motor de cliente web de código abierto llamado "Chromium". Lo podemos instalar en Linux, Android, iOS, Windows i macOS.
- **Microsoft Edge**: el nuevo navegador web desarrollado por Microsoft con el objetivo de sustituir al veterano "Internet Explorer" del que se ha anunciado que su versión 11 será la última. Lo podemos instalar en Android, iOS, Windows, macOS, xBox One.
- **Mozilla Firefox:** es el navegador open-source más utilizado hoy en día. Creado y mantenido por la actual comunidad Mozilla Foundation como el sucesor espiritual del navegador Netscape Navigator. Fue uno de los mayores competidores de Internet Explorer hasta la aparición de Chrome. Los principales sistemas operativos en los que lo podemos instalar son: Linux, Android, iOS, Windows, macOS.

El navegador ha de ser capaz de comunicarse con el cliente, mostrar la página web, y recibir las interacciones del usuario. Para ello las tecnologías más relevantes utilizadas son:

- HTML (Hype Text Markup Language): es el lenguaje de marcas utilizado para estructurar páginas web con texto y contenido multimedia.
- CSS (Cascade Style Sheets): es el lenguaje utilizado para dar estilos a la estructura HTML y su contenido.

El navegador ha de ser capaz también de interpretar lenguajes de programación en entorno cliente. No todos los navegadores vienen con todos los intérpretes para todos los lenguajes de programación en entorno cliente. Por ello en ocasiones el navegador nos pedirá instalar intérpretes específicos. Las principales tecnologías de programación en entorno cliente son:

- JavaScript: es el lenguaje de programación más utilizado en la capa de cliente. Desarrollado inicialmente por Netscape bajo el nombre de Mocha posteriormente se modificó por JavaScript cuando aparecieron los Applets de Java. Nos permite modificar la página web y facilitar la interacción con el usuario. Actualmente también lo podemos encontrar en el lado de servidor o en aplicaciones de escritorio. Este lenguaje es el que utilizaremos durante todo el módulo.
- Adobe Flash: es una tecnología propiedad de Adobe que permite crear contenido interactivo y multimedia muy potente. Podemos programar los contenidos multimedia creados con Flash mediante el lenguaje de programación específico "Adobe Action Script". Actualmente el uso de Flash en la integración web es más bien reducido debido en gran parte a que es una tecnología propietaria, la poca integración con el resto del contenido HTML, la dificultad de los buscadores para indexar sus contenidos y a las nuevas versiones de HTML, CSS y JS que facilitan la creación de animaciones en la web.
- Visual Basic Script (ActiveX): es el lenguaje de programación web que implementó Microsoft en un intento de competir con JavaScript. Al ser un lenguaje propietario de Microsoft la incompatibilidad con otros navegadores y el gran auge de JavaScript lo han dejado en desuso.
- Applets de Java: es una manera de poder ejecutar programas Java con interfaz visual desde nuestro navegador. Son tratados como programas independientes a la página web, por lo que no tienen acceso a ella. Para poder ser ejecutados es necesario que el usuario tenga instalada la máquina virtual de java.

1.1.4.2. Capacidades y limitaciones en entorno cliente

El hecho de poder ejecutar código directamente en el Navegador implica ciertas características comunes aunque puede tener matices para algún lenguaje en concreto.

- El navegador ha de contener el intérprete para poder ejecutar el código, sino no se podrá ejecutar.
  En el caso de Java, VBScript y Flash solo hay un único intérprete oficial que es mantenido y gestionado por su respectiva entidad oficial. Pero en el caso de JavaScript no existe una entidad única encargada de desarrollar su intérprete sino que cada navegador puede desarrollar su propio intérprete a partir de la especificación EMA Script que indica cómo debería de funcionar un código JavaScript. Esto conlleva a que distintos navegadores pueden ejecutar de

distinta forma el mismo código (o incluso no ejecutar alguna sentencia), aunque por suerte cada vez las diferencias son menores.

- El acceso a los recursos del ordenador del cliente viene limitado por el intérprete. Si en Java tenemos pleno acceso a los recursos del navegador pudiendo crear y borrar archivos por todo el disco duro, por razones de seguridad los lenguajes ejecutados en un navegador lo deben hacer bajo estrictas medidas de seguridad para evitar acciones indeseadas por el usuario.
- Pueden hacer peticiones al servidor para obtener nuevos datos. Ésta comunicación se suele hacer sin necesidad de refrescar el navegador, de ésta forma la experiencia del usuario es mucho más agradable. Cuando trabajemos con JavaScript veremos la tecnología AJAX para realizar éste tipo de peticiones.
- Tanto JavaScript, VBScript, Adobe Action Script son lenguajes llamados de "script" o lenguajes "interpretados". Los programas desarrollados con estos lenguajes no son compilados antes de ser ejecutados, sino que sus intérpretes los ejecutan a partir del código fuente.
- Tanto JavaScript como VBScript puede acceder y modificar la estructura HTML de la web y sus estilos. De ésta forma podremos alterar la interfaz gráfica del usuario sin tener que recargar la página, haciendo que elementos HTML cambien de tamaño, color o contenido. También podremos gestionar la interacción del usuario con la web, controlando sus acciones con el ratón, el teclado o la interacción con el resto de elementos de la web como el control de formularios.

1.1.4.3. Principales tecnologías en la capa de servidor

Siempre que hablamos de páginas web podemos diferenciar entre aquellas con un contenido "estático" o aquellas con un contenido "dinámico".

Las páginas web de contenido estático son aquellas definidas en un archivo ".html". Como HTML es un lenguaje de estructuración y no de programación, el servidor no ejecuta el archivo ".html" sino que simplemente lo retorna al navegador. Es por ello que el contenido que va a mostrar un navegador cada vez que pida el mismo archivo ".html" siempre va a ser el mismo.

Las páginas web de contenido dinámico también generan como resultado un código HTML, pero éste resultado puede variar cada vez que se pida la página según los parámetros de la petición. Esto se debe a que cada vez que se pida un contenido dinámico el servidor va a ejecutar el archivo que contenga el código correspondiente a ése contenido para generar la respuesta con el código HTML.

Aunque hay muchos lenguajes de programación orientados a generar contenidos web dinámicos y muchos programas capaces de hacer de servidor, no todos los servidores son capaces de interpretar todos los lenguajes de programación. Por ello cuando debamos escoger un lenguaje de programación deberemos tener presente el servidor que necesitaremos y viceversa.

A continuación veremos los principales lenguajes que podemos encontrar en la capa de servidor y posteriormente veremos los principales programas que podemos encontrar como servidores:

- **PHP**: es un lenguaje de programación Open Source utilizado para generar contenido web de forma dinámica. Suele ser utilizado para generar código HTML que el cliente interpretará como una página web, aunque también puede generar otros tipos de archivos como imágenes, vídeos, .pdf , etc.. Aunque menos utilizado, también se puede utilizar para generar aplicaciones de escritorio en entorno gráfico o por línea de comandos.
  El bajo coste de mantenimiento de un servidor PHP y su rápida curva de aprendizaje lo ha convertido en uno de los lenguajes de programación más utilizados en servidores web para pequeños proyectos.

  Los archivos que contiene código PHP se identifican con la extensión ".php".

- **JSP** (Java Server Pages): es una tecnología que permite generar páginas web ejecutando código Java, por lo que la convierte en una tecnología muy potente y robusta. Suele ser un lenguaje de programación en entorno cliente utilizado por proyectos grandes, ya que el mantenimiento de un servidor web Java es más costoso que un servidor web PHP y además es una tecnología con una curva de aprendizaje más lenta que PHP.
  Los archivos que contienen código JSP se identifican con la extensión ".jsp".

- **ASP.NET**: es un framework de Microsoft que facilita la creación de aplicaciones web dinámicas e interactivas ofreciendo un conjunto de soluciones predefinidas y personalizables para ajustarse a las distintas necesidades del mundo web. Su principal estandarte es la robustez y seguridad al tener el respaldo de una empresa como Microsoft. Por ello su principal desventaja es el alto coste de sus servidores y licencias para su desarrollo.
  Los archivos que contienen código ASP se identifican con la extensión ".asp".

Como hemos visto, los programas que se encargan de recibir las peticiones del cliente y ofrecerles una respuesta son los llamados "servidores". A continuación veremos los principales servidores que podemos encontrar:

- **Servidor HTTP Apache**: servidor de código abierto mantenido por una comunidad de usuarios bajo la supervisión de Apache Software Foundation. Su gran flexibilidad y modularidad hace que se pueda configurar para trabajar junto a un gran conjunto de lenguajes y tecnologías entre las que podemos destacar: PHP, Perl, Python o Ruby . Es uno de los servidores web más populares y utilizados por su alta flexibilidad y su bajo coste de mantenimiento. Lo podemos instalar en cualquier máquina Windows o Linux.
- **Tomcat**: servidor de código abierto mantenido por la Apache Software Foundation capaz de generar páginas JSP. Puede trabajar de forma autónoma como servidor web o junto a un servidor Http Apache (quien recibirá las peticiones y las enviará a Tomcat para que genere la respuesta). Lo podemos instalar en cualquier máquina Windows o Linux que disponga de una máquina virtual Java. Es un servidor con un coste de mantenimiento un poco más elevado que Http Apache.
- **IIS**: servidor propiedad de Microsoft capaz de generar páginas ASP. Es un servidor muy modular al que se le pueden ir añadiendo muchas extensiones para añadir nuevos servicios. Como solo lo podemos instalar en Windows NT y hay que pagar las licencias de Microsoft, su coste es más elevado.

Por su notoriedad actual haremos mención especial a NodeJS.

- **NodeJS**: no es específicamente un servidor web, sino un entorno de ejecución basado en JavaScript creado con la idea de poder ejecutar servidores web altamente escalables. Es decir, NodeJS lo podemos entender como la máquina virtual de Java pero para JavaScript.

1.1.5. Herramientas

Podemos programar en JavaScript con cualquier IDE o incluso con el bloc de notas, pero veremos el IDE Visual Studio Code, un IDE muy ligero, práctico y versátil.

Como trabajaremos con código JavaScript ejecutado por el navegador deberemos crear un documento .html al que le vincularemos el archivo JavaScript con extensión .js. Cuando visualicemos el HTML el navegador será el encargado de ejecutar los JavaScript asociados.

Visual Studio Code no permite de forma nativa visualizar un documento HTML, pero tiene muchos plugins para hacerlo y nosotros utilizaremos el plugin Live Server. Los plugins se pueden configurar editando un archivo JSON asociado.

Como JavaScript es un lenguaje interpretado, los errores que se pueden detectar en el mismo IDE son escasos. Por ello la mayor parte de la depuración lo realizaremos en el mismo navegador. La mayoría de navegadores tienen herramientas de inspección de código para depurar el código JavaScript así como el código HTML , CSS, etc.. pero utilizaremos el Chrome por sus buenas herramientas de desarrollo.

Una vez inspeccionemos el código podremos acceder a la consola JavaScript en donde aparecerán todos los errores, al inspector de scripts en donde podremos ver el código y poner breakpoints y al inspector de red en donde veremos las peticiones web que se realizan.

En el siguiente video podemos ver como instalar, configurar y utilizar Visual Studio Code y Google Chrome para el desarrollo web en JavaScript

Video: Instalación y configuración del IDE

1.2. Introducción a JavaScript

JavaScript es el lenguaje de programación más utilizado en la capa de cliente. Desarrollado en 1995 por Netscape bajo el nombre de Mocha finalmente se renombro a JavaScript a causa de la irrupción del lenguaje Java en el mundo web con la aparición de su tecnología de Applets. A partir de JavaScript se definió la especificación ECMA Script, como una base a partir de la cual se pudieran desarrolla distintos lenguajes de programación. En principio cualquier intérprete de ECMA Script debería ser capaz de ejecutar la mayoría de funciones de JavaScript.

Hay que remarcar que Java y JavaScript son dos lenguajes distintos que comparten muy pocas similitudes. Sin duda JavaScript es el lenguaje de programación más utilizado actualmente interpretado por navegadores web.

A partir de este punto siempre que nos refiramos a JavaScript lo haremos pensando en entorno cliente. Y es en éste entorno donde su objetivo siempre ha de ser facilitar la interacción de la web con el usuario, es decir, mejorar su usabilidad y su accesibilidad. Y por ello todos los códigos JavaScript siempre van a ir vinculados a un documento HTML. Cada documento HTML puede tener vinculados múltiples códigos JavaScript y un código JavaScript puede ser vinculado desde múltiples documentos HTML.

Como JavaScript es un lenguaje interpretado, los códigos fuente de los programas no son compilados antes de que los reciba el cliente. Esto implica que cuando un documento HTML tiene vinculado un documento JavaScript, el navegador pide el documento al servidor quien le retorna el código fuente. El navegador leerá el código y lo añadirá a su intérprete de JavaScript para que lo ejecute cuando el código lo indique. Hay que destacar que como usuarios siempre vamos a poder leer los códigos JS que ejecute nuestro navegador.

Por ello, teniendo presente su objetivo y sus características, JavaScript nunca ha de ser el lenguaje escogido para mejorar la seguridad de una web. Todas las validaciones y comprobaciones de seguridad se han de realizar siempre en el servidor. Y es que en JavaScript sí que debemos hacer validaciones y comprobaciones, pero nunca como medida de seguridad sino para mostrar mensajes personalizaos de error al usuario y así mejorar su experiencia.

En concreto veremos que JavaScript nos permite modificar la visualización de la página web alterando las propiedades de los elementos HTML, mostrar animaciones 2D/3D, enviar y recibir peticiones al servidor y en general controlar todas las interacciones con el usuario.

A continuación veremos cómo podemos vincular un documento HTML con un documento JavaScript.

1.2.1. Integración en el HTML

Podemos vincular un código JavaScript a un documento HTML de tres maneras distintas.

La primera manera es abriendo una etiqueta <script> dentro del código HTML y añadiendo el código JavaScript dentro. Nos quedaría entonces el código HTML de la siguiente manera:

Ejemplo.html

```
<!DOCTYPE html>
<head>
    <title>Document</title>
    <script>
    // código JavaScript
    </script>
</head>
<body>
```

Código JavaScript escrito directamente dentro un documento HTML.

Esta manera que hemos visto es muy fácil e intuitiva pero tiene el hándicap de que si queremos reaprovechar ese código JavaScript para otra página web lo tendremos que duplicar, haciendo inviable su escalabilidad y mantenimiento. Para evitar éstos problemas vamos a ver otra manera.

La segunda manera es tener el código JavaScript ubicado en un documento a parte con extensión .js y vincularlo con el HTML mediante la etiqueta <script>, pero en este caso le añadiremos a la etiqueta un atributo *src* que contendrá la ruta relativa desde el propio documento HTML hasta el documento JavaScript.

Ejemplo.html

```
<!DOCTYPE html>
<head>
    <title>Documento HTML</title>
    <script src="ejemplo.js"></script>
</head>
<body>

</body>
```

ejemplo.js

```
//aquí el código JavaScript
```

Ejemplo de cómo vincular un código JavaScript

La tercera manera consiste en vincular dentro del mismo código HTML un código JavaScript a una cierta acción realizada sobre un elemento HTML, lo que veremos más a delante como “eventos”. Para crear éste vínculo en el mismo HTML hemos de añadir al elemento sobre el que vamos a realizar la acción un atributo que indique qué acción realizaremos sobre él. A este atributo le daremos como valor el código JavaScript que querremos que se ejecute al realizarse la acción.

En el siguiente ejemplo el código JavaScript se ejecutará cuando el usuario clique encima del texto “Titulo”.

```
<body>
    <h1 onclick=" //aquí el código Javascript">
        Titulo
    </h1>
</body>
```

La manera más utilizada y más recomendable es la segunda, ya que resulta la más controlable, escalable y reutilizable. Pero ello es la forma que utilizaremos en éste módulo.

Una vez vinculado el código JavaScript con el documento HTML seguramente nos asalten dos dudas: “pero como se ejecuta? “ y “importa si lo vinculo en el <body>?”. Las dos dudas van relacionadas y las resolveremos en el siguiente punto.

1.2.2. Ejecutando el JavaScript

Podemos añadir el código JavaScript en cualquier posición del documento HTML que no lo convierta en un HTML inválido (no lo podemos añadir fuera de <html> o entre </head> y <body>, etc..).

El código JavaScript va a ser ejecutado por el intérprete JavaScript del navegador en el momento en que se carga. Como los navegadores por defecto cargan su contenido de forma secuencial empezando por arriba, si en el código JavaScript hemos programado una operación matemática, esta se ejecutará cuando se cargue el JavaScript.

También vamos a poder añadir distintos códigos JavaScript en distintos puntos de nuestro código HTML como podemos ver en el siguiente ejemplo:

```
<!DOCTYPE html>
<head>
    <title>Document</title>
    <script> //aquí el 1r código </script>
</head>
<body>
    <script src="ejemplo2.js"></script>
    <h1 onclick=" //aquí el 3r código ">
        Titulo
    </h1>
    <script> //aquí el 4o código </script>
</body>
</html>
```

Ejemplo con 3 códigos JavaScript ubicados en distintas posiciones dentro de un código HTML.

En el ejemplo anterior, el primer código a ejecutar será el "1r código", el segundo será "ejemplo2.js" y el tercero será "el 4º código". El "3r código" será ejecutado solo cuando el usuario clique sobre el elemento H1 (es decir, el texto Titulo).

Es importante remarcar que si el "1r código" necesita de las funciones definidas en el "4º código" para ejecutar código mientras se carga la web, primero se deberá vincular el "4º código" y luego el "1r código".

Pero aunque podamos añadir la etiqueta <script> en cualquier posición dentro de nuestro HML, teóricamente se recomienda añadirla dentro del elemento <head> del HTML, ya que el código JavaScript aporta meta-información de la página web. Pero a la práctica veremos que en muchas ocasiones se nos recomienda añadir los archivos JavaScript justo antes de la etiqueta </body> por dos razones principales:

- La primera razón es que para una buena experiencia del usuario es prioritario que se cargue todo el texto y las imágenes antes que todos los códigos JavaScript.
- La segunda razón es que si situamos el JavaScript antes del <body> e intentamos modificar el contenido HTML obtendremos un error, ya que el contenido HTML dentro del <body> aún no se habrá cargado. Insertar el código JavaScript después de todo el HTLM nos asegura que el navegador ya habrá cargado los elementos HTML anteriores y evitaremos el error de acceder a un elemento aún no cargado.
  Hay básicamente dos formas más elegantes de solucionar este problema:
  - La primera (que veremos más a delante) es indicar mediante JavaScript que se ejecute el código una vez se haya cargado toda la página web.

- La segunda es añadir a la etiqueta <script> el atributo ***defer***, lo que obliga al navegador a ir descargando el resto de código HTML a la vez que se descarga el JavaScript pero solo se ejecutar el script cuando el documento html hay terminado de cargarse. Pero ésta segunda manera solo funciona con scripts externos al HTML.

1.2.3. Sintaxis

JavaScript es un lenguaje case-sensitive y por lo tanto diferencia entre mayúsculas y minúsculas. Así que si definimos una variable o función en mayúsculas la deberemos escribir también en mayúsculas para hacer referencia a ella.

En JavaScript se suele terminar cada sentencia con un punto y coma ";" o con un salto de línea. Aunque separar cada sentencia con puno y coma es recomendable sobretodo sabemos que vamos a querer comprimir el código, ya que una de las acciones principales para comprimir el código es eliminar todos los satos de línea, así que si no hemos puesto puntos y comas las sentencias no quedarían separadas y no se podrían distinguir.

Los nombres que demos a las variables o funciones han de empezar por una letra o el guion bajo "_", aunque se recomienda empezar siempre en minúsculas y reservar el guion bajo para ciertos constructores JavaScript.

Cuando una variable o función contenga un nombre compuesto, JavaScript se suele utilizar la convención "lowe camel case", que consiste en empezar cada palabra concadenada con mayúsculas. Es decir, si queremos crear una variable "mi nombre", escribiremos como nombre de variable "miNombre".

Por último vamos a ver el listado de palabras reservadas que no se pueden utilizar como nombres de variables o funciones:

- reak
- case
- catch
- class
- const
- continue
- debugger
- default
- delete
- do
- else
- export
- extends
- finally
- for
- function
- if
- import
- in
- instanceof
- new
- return
- super
- switch
- this
- throw
- try
- typeof
- var
- void
- while
- with
- yield

1.2.4. Comentarios

Los comentarios son textos que podemos escribir en el código pero que no se ejecutan. Su función es describir qué hace nuestro programa para que sea fácilmente entendible.

En JavaScript podemos escribir los comentarios de dos formas distintas:

- Una línea: el código a la derecha de las dos barras "//" nunca se va a interpretar

  Por ejemplo: //esto es un comentario de una línea

- Múltiples líneas: el código entre la barra y asterisco "/*" y el asterisco y barra "*/" nunca se va a interpretar aunque contenga múltiples líneas

  Por ejemplo: /* esto es un comentario

  De múltiples líneas

  */

1.2.5. Declaración de variables

En JavaScript podemos declarar una variable con la palabra clave *var* o *let* seguido del nombre que queramos dar a la variable. No indicaremos el tipo de dato que almacenará la variable ya que en distintos momentos flujo de ejecución vamos a poder asignar otros tipos de datos a la variable. Esta propiedad es la que llamaremos como "no tipado" y diremos que JavaScript es un lenguaje de programación "no tipado" porque una variable puede almacenar distintos tipos de variables a lo largo de la ejecución del programa.

Cuando hemos declarado una variable pero aún no le hemos asignado ningún valor, ésta contendrá el valor *undefined*. Por otro lado, si intentamos acceder a una variable no definida obtendremos un error.

Las últimas versiones de JavaScript permiten declarar una variable como una constante utilizando la palabra clave *const.* Cuando declaramos una variable con *const* su valor no va a poder ser cambiado.

Una vez hemos declarada una variable le podemos asignar un valor con el operador " = " seguido del valor que le queramos dar. Cuando creamos una variable sin utilizar un constructor diremos que estamos creando un "*Literal*".

Por ejemplo:

```
var nombre = "Jack"; //variable literal String
let apellido = new String("Mewto"); //variable objeto tipo String
var edad = 25; //variable literal Number
const PI = new Number(3.14); //variable objeto tipo  Number
```

En este ejemplo hemos declarado 3 variables, dos de tipo String y una de tipo Number.

Como podemos observar en el ejemplo, si utilizamos literales en ningún momento hemos de indicar el tipo de dato de la variable, sino que el tipo de dato se modifica automáticamente para poder contener el valor asignado.

También podemos observar cómo hemos utilizado la palabra clave *let* y *var* indistintamente para declarar las variables, pero tienen dos diferencias:

- La primera es que el ámbito de las variables declaradas con *let* se limita a el bloque de código en el que se ha definido, mientras que las variables definidas con *var* siempre son globales. Esto lo veremos con ejemplos al ver las estructuras de control de flujo.
- La segunda viene derivada de la primera, y es que una variable declarada con *let* no puede volver a ser declarada dentro del mismo bloque de código. Mientras que un variable declarada con *var* puede ser declarada tantas veces como se quiera.

Como el uso de *let* es más restrictivo que *var*, en general se recomiendo utilizar la palabra clave *let* para declarar variables.

1.2.6. Uso de strict

Por defecto en JavaScript podemos asignar un valor a una variable sin previamente haberla definida con *var* o *let*. Esto puede provocar que errores de ortografía al utilizar variables pasen desapercibido por el IDE y el navegador y sean interpretados como una nueva variable haciendo que el programa funcione incorrectamente. Podemos forzar a JavaScript a que nos indique esos errores añadiendo la sentencia *"use strict";* al inicio de nuestro código JavaScript.

En el siguiente ejemplo asignamos un valor a una variable nombre que no ha sido declarada con *let* o *var* produciendo un error.

```
"use strict";
nombre = "Jack"; //saltarà un error al no estar declarado
let apellido = new String("Mewto"); //variable objeto tipo String
```

Ejemplo uso de “use strict”.

1.2.7. Tipo de variables

A continuación veremos los principales tipos predefinidos de variables que nos ofrece JavaScript.

tipo	ejemplo	Descripción
Number	let phCafe =4.5; var edad = 25;	Permite almacenar números enteros o decimales separados por puntos. Si escribimos el número entre comillas JavaScript lo interpretará como una String. Este es el único tipo de variable numérica del que dispone JavaScript.
String	let nombre="Juan"; let apellido='Gómez'; let parrafo=`texto multi linea`; let nombreCompleto = nombre + apellido;	Permite almacenar un texto en. Cuando queramos almacenar un texto en una variable escribiremos el texto entre comillas dobles o comillas simples. Para poder introducir un salto de línea como contenido de una String, utilizaremos como comillas el acento hacia la izquierda “ ` ” . Podemos concadenar dos Strings utilizando el operador “+” . Cualquier variable concadenada con una String es convertida a String
Boolean	let esDeDia=true;	Permite almacenar dos únicos valores: *true* o *false*. Es importante remarcar que el valor no se ha de indicar nunca entre comillas. *true* y *false* son un valor en sí mismo.
Array	let notas=[3.4,7.7, 9];	Permite almacenar múltiples valores en una misma variable. Este tipo de variables los veremos con más profundidad más a delante.

Tabla con los principales tipos predefinidos de variables

La variable Number además de valores decimales en JavaScript también puede almacenar valores binarios, octales y hexadecimales utilizando la siguiente notación:

- Binarios: empezando el número con *0b* seguido de los ceros y unos que lo formen. Por ejemplo: *let n= 0b101*; para representar el 5
- Octales: empezando el número con un *0* seguido de los números entre 0 y 7 que lo formen. Por ejemplo: *let n = 017*; para representar el 15
- Hexadecimal: empezando el número con un *0x* seguido de los números entre 0 y 9 y las letras entre A y F que lo formen. Por ejemplo: *let n=0xA*; para representar el 10.

1.2.8. Conversión entre tipos de variables

Aunque javascript sea un lenguaje de programación "no tipado" en muchas ocasiones necesitaremos almacenar un dato de un determinado tipo, por ejemplo pasando de numérico a string o a la inversa. Vamos a ver un ejemplo de tres conversiones básicas que nos ofrece JavaScript

- Para convertir una variable a string es suficiente con concadenar la variable que quedamos convertir con una cadena vacía.
- Para convertir una variable a un valor numérico sin decimales basta con utilizar la función parseInt.
- Para convertir una variable a un valor numérico con decimales basta con utilizar la función parseFloat.

En el siguiente código vamos a ver un ejemplo de conversión primero de numérico a String, luego de String a entero y finalmente de String a decimal:

```
let texto=""+123.55;  //texto vale "123.55"
let entero= parseInt(texto); //entero vale 123
let decimal = parseFloat(texto); //decimal vale 123.55
```

Ejemplo de conversiones entre distintos tipos de datos

1.2.9. Operadores

Los operadores son marcas que nos permiten operar con una o varias variables.

A continuación veremos los operadores aritméticos enfocados a realizar operaciones matemáticas con las variables:

1.2.9.1. Operadores aritméticos

Los operadores aritméticos nos permiten operar con las variables. JavaScript dispone los más comunes que podemos encontrar en la mayoría de lenguajes de programación:

Operador	Uso	Descripción
+	let resul = 5 + 10;	*resul* guarda el valor 15 como resultado de sumar 5 más 10
-	let resul = A - B;	*resul* guarda el resultado de restar a la variable *A* el valor de la variable *B*
*	let resul = 5 * A;	*resul* guarda el resultado de multiplicar 5 por el valor de la variable *A*
/	let resul = A / 5;	*resul* guarda el resultado de dividir por 5 el valor de la variable *A*
%	let resul = 10 % 2;	*resul* guarda el 10 en módulo 2 (es decir 0). Utilizado comúnmente para conocer si un número es múltiple de otro

Tabla de los distintos operadores aritméticos

1.2.9.2. **Operadores unarios**

Los operadores unarios son aquellos que solo necesitan de un operando.

Operador	Uso	Descripción

typeof	let numero = 33;
	let marca="Naiki";
	typeof numero; //"number"
	typeof marca; //"string"
		Este operador nos retorna una String indicando el tipo de variable que precede. En este caso: *typeof numero;* retorna "number" y *typeof marca;* retorna "string".

delete	miObjeto.propiedad; arrayAssoc["posición"];	Retorna true si puede eliminar la propiedad o el valor ubicado en una cierta posición en un array asociativo.
%	let resul = 10 % 2;	*resul* guarda el 10 en módulo 2 (es decir 0) . Utilizado comúnmente para conocer si un número es múltiple de otro.
++	let resul = 5++;	Incrementa en 1 el valor. La variable *resul* contendrá el valor 6 como resultado de sumar 5 + 1.
--	let resul = A--;	Decrementa en 1 el valor. La variable *resul* contendrá el valor de A menos 1 como resultado de restar 1 al valor de la variable A
-	let resul = - A;	La variable resul contendrá el resultado de cambiar de signo el valor que contenga A
!	let resul = !false;	Alterna el valor booleano(true o false) que precede. La variable resul contendrá el valor *true*.

Tabla de los distintos operadores unarios

1.2.9.3. Operadores de asignación

Los operadores de asignación nos permiten asignar un valor a una variable. JavaScript dispone los más comunes que podemos encontrar en la mayoría de lenguajes de programación:

Operador	Uso	Descripción
=	**let resul = 10;**	Asigna a la variable de su izquierda el valor que resulte de la operación situada a la derecha. resul contendrá el valor 10.
+=	let resul = 10; **resul += 5;**	*resul* contendrá 15 como resultado de sumarse 5 a ella misma.
-=	let resul = 10; **resul -=6;**	*resul* contendrá -4 como resultado de restarse 6 a ella misma.
*=	let resul = 10; **resul *= 6;**	*resul* contendrá 60 como resultado de multiplicar su valor por 6.
/=	let resul = 10; **resul /= 5;**	*resul* contendrá 2 como resultado de dividir su valor por 5.
%=	let resul = 10; **resul %= 5;**	*resul* contendrá 0 como resultado de calcular 10 en módulo 5.

Tabla de los distintos operadores aritméticos de asignación

1.2.9.4. **Operadores relacionales**

Los operadores de asignación nos permiten comparar dos valores para saber si son iguales o cual es mayor y cual menor. JavaScript dispone los más comunes que podemos encontrar en la mayoría de lenguajes de programación además de su particular operador de comparación triple:

Para la siguiente tabla supondremos que la variable A tiene un valor de 10.

Operador	Uso	Descripción
==	let A = 10; **(A==10)** let B = "10"; **(B==10)**	Evalúa si A tiene un valor de 10. El resultado será ***true*** porque A es igual a 10 en valor. Evalúa si B tiene un valor de "10". El resultado será ***true*** porque B es igual a "10" en valor (aunque sean datos de distinto tipo).
===	let A = 10; **(A===10)** let B = "10"; **(B===10)**	Este operador es el llamado "igual estricto", y comprueba si dos valores son iguales en valor y tipo de datos (si son enteros, strings, etc..). Evalúa si A es igual en valor y tipo que 10. El resultado será ***true*** porque A es igual a 10 en valor y en tipo. Evalúa si B es igual en valor y tipo que "10". El resultado será ***false*** porque B es igual a "10" en valor pero son datos de distinto tipo.
!=	let A = 10; **(A!=10)** let B = "10"; **(B!=10)**	Evalúa si A tiene un valor distinto a 10. El resultado será ***false*** porque A es igual a 10 en valor. Evalúa si B tiene un valor distinto a "10". El resultado será ***false*** porque B es igual a "10" en valor.

!==	let A = 10; **(A!=10)** let B = "10"; **(B!==10)**	Este operador es el llamado “distinto estricto”, y comprueba si dos valores son distintos en valor y tipo de datos. Evalúa si A tiene un valor distinto a 10. El resultado será ***false*** porque A es distinto a 10 en valor. Evalúa si B tiene un valor distinto a “10”. El resultado será ***true*** porque B es distinto a “10” en tipo.
>	let A = 10; (A > 10)	Evalúa si A tiene un valor superior a 10. El resultado será ***false*** porque A no tiene un valor superior a 10.
<	let A = 10; (A < 15)	Evalúa si A tiene un valor inferior a 15. El resultado será ***true*** porque A es inferior a 10.
>=	let A = 10; (A >= 10)	Evalúa si A tiene un valor igual o superior a 10. El resultado será ***true*** porque A es igual a 10.
<=	let A = 10; (A <= 5)	Evalúa si A tiene un valor inferior o igual a 5. El resultado será ***false*** porque A no es inferior ni igual a 5.

Tabla de los distintos operadores relacionales

1.2.9.5. **Operadores Lógicos**

Nos permiten realizar operaciones lógicas obteniendo un valor boolean (true o false) a partir de dos valores boolean. Si utilizamos valores no numéricos nos retornará el valor de una de las variables según se describe en la siguiente tabla:

Operador	Uso	Descripción
&&	(A && B)	**A and B**. Utilizado con variables booleanas el resultado será true solamente si A y B son *true*. En el resto de casos el resultado será false. Utilizado con variables no booleanas, retornará A si se puede convertir a *false*.
||	(A || B)	**A or B.** Utilizado con variables booleanas el resultado será *true* si A o B o ambos son *true*. Solo cuando ambos sean *false* el resultado será *false*. Utilizado con variables no booleanas, retornará A si se puede convertir a *true*.
!	(!A)	**Not A**. Utilizado con variables booleanas, en el caso que A sea true el resultado será false, si A es false el resultado será true. Utilizado con una variable no booleana, retornará A si se puede convertir a *true*.

Tabla de los distintos operadores lógicos

1.2.9.6. Otros operadores

Los operadores ternarios son aquellos formados por 3 operados. En JavaScript solo tenemos un único operador ternario: el operador condicional.

Operador	Uso	Descripción
? :	(10>5) ? "si": "no"	**Operador condicional**. Nos permite indicar dos posibles valores a retornar según una comparación. Antes del *?* se indica la comparación, después se indica el valor a retornar en caso que la comparación sea correcta y después de los *:* el valor a retornar si la comparación es incorrecta.
instanceof	new String("p") instanceof String	Indica si un valor es una instancia de un tipo de objeto.

Tabla de los distintos operadores ternarios de JavaScript

1.2.10. Estructuras de control de flujo

Las estructuras de control nos permiten alterar el flujo de ejecución de un programa para que un bloque de código pueda o no ejecutarse, se repita múltiples veces, deje de ejecutarse o se ejecute solo si se detecta algún error.

JavaScript incorpora las principales estructuras de control de flujo que podemos encontrar en la mayoría de lenguajes de programación.

1.2.10.1. If/Else

La sentencia *if(condición){ … }* nos va a permitir indicar una expresión booleana y un bloque de código a evaluar si el resultado de la expresión es *true.* En el caso que el resultado sea *false* se evaluará el bloque de código definido con la sentencia *else{ }*. La sentencia *else* es opcional, pero si la queremos añadir siempre ha de ir inmediatamente después del bloque de código definido por el *if.*

Podemos anidar estructuras de selección sin problema teniendo presente que las variables definidas con *let* solo están disponibles dentro de los bloques de código en los que se han definido. En el siguiente ejemplo se mostrará una alerta al usuario con el mensaje "Es igual a 4!"

```
let numero = 4;
if (numero == 3) {
   alert("Es igual a 3!");
} else {
   if (numero == 4) {
      alert("Es igual a 4!");
   }
}
```

Código de ejemplo de una estructura if/else

1.2.10.2. **Switch / case**

La estructura Switch/Case nos permite buscar una coincidencia entre una variable y distintos valores. Las coincidencias las indicaremos con la palabra clave *case* seguido del valor que queramos comparar. A cada *case* le podremos indicar el código a ejecutar en el caso de que produzca la coincidencia. Después de ejecutar el contenido de uno de los *case* automáticamente se ejecutará el código del siguiente *case* a no ser que se encuentre con un *break;*
La sintaxis se compone de:

- *switch(valorComparar){ }* en donde se define la variable a comparar. Limita el bloque de código que contendrá los distintos casos a comparar.
- *case opcion1:* contiene el código a ejecutar cuando el *valorCompara* sea igual a *opcion1*.
- *break;* sentencia opcional que evita se siga ejecutando el contenido del siguiente case.
- *default:* contiene el código a ejecutar cuando no se cumpla ningún *case*.

El siguiente ejemplo mostrará el mensaje “Hola !” y después el mensaje “Adios !”. Si la variable *opción* tuviera el valor “animar” mostraría los valores “Ánimos” y “mensaje por defecto”.

```
let opcion = "saludar";
switch (opcion) {
   case "saludar":
      alert("Hola !");


   case "despedir":
      alert("Adios !");
      break;
   case "animar":
      alert("Ánimos !");

   default:
      alert("mensaje por defecto");
}
```

Código de ejemplo de una estructura switch/Case

1.2.10.3. **while**

La estructura *while(condicion){}* permite definir un bloque de código a ejecutar mientras se cumpla una condición.
Entre los paréntesis indicaremos la condición que se ha de cumplir para ejecutar el bloque definido entre las llaves. Una vez ejecutado se volverá a evaluar la condición de nuevo para ver si se tiene que volver a ejecutar. Éste proceso se repetirá mientras que la condición siga siendo válida.

El siguiente ejemplo mostrará tres alertas con los siguientes mensajes:

El valor de número es 0

El valor de número es 1

El valor de número es 2

```
let numero=0;
while(numero < 3){
   alert("El valor de número es:"+numero);
   numero++;
}
```

Código de ejemplo de una estructura while(){ }

1.2.10.4. **for**

La estructura *for (contador; condició ; modificación del contador){ …. }* permite definir un bloque código a ejecutar mientras se cumpla una condición.
A diferencia del w*hile*, el *for* permite iniciar la variable que utilizaremos para controlar el bucle, definir la condición para volver a repetir el bloque de código y escribir la operación que altere la condición para finalizar el bucle.
Si variable usada para controlar el bucle se define dentro del mismo *for* con un *let*, no estará disponible una vez finalizado el bucle.

El siguiente ejemplo mostrará tres alertas con los siguientes mensajes:

El valor de número es 0

El valor de número es 1

El valor de número es 2

```
let numero=0;
while(numero < 3){
   alert("El valor de número es:"+numero);
   numero++;
}
```

Código de ejemplo de una estructura for

1.2.10.5. **do while**

La estructura *do{ }(condición)while;* permite definir un bloque código a ejecutar mientras se cumpla una condición.
A diferencia del while, el *do while* evalúa la condición después de haber ejecutado el bloque de código, lo que implica que el bloque de código que contiene como mínimo se ejecute una vez.
El siguiente ejemplo mostrará una alerta con el mensaje:

El valor de número es 0

```
let numero=0;
do{
//bloque de código a repetir
   alert("El valor de número es:"+numero);
}while(numero != 0);
```

Código de ejemplo de una estructura do while

1.2.10.6. try /catch /finally

La estructura *try{ } catch(infoExcepcion){ } finally{ }* nos permite programar las acciones a realizar si se produce algún error durante la ejecución de un código. Está formado por 3 partes, *el try{}* el *catch(){}* y *el finally{}:*

- *try{}* : contiene el bloque de código del que querremos controlar los posibles errores que se produzcan al ejecutar el programa
- *catch(variable){}* : contendrá las acciones a realizar cuando se produzca un error
- *finally{} :* parte opcional que contendrá las acciones que siempre querremos realizar aunque se haya producido o no un error.

El siguiente ejemplo mostrará dos alertas con los mensajes:

Error: ReferenceError: y is not defined

Cuidado con no declarar las variables

```
let x=20, r;
try {
   r= x/y;
   alert(r); //ésta alerta nunca se lanza

}
catch (err) {
  alert("Error: " + err);

} finally {
  alert("Cuidado con no declarar las variables") ;
}
```

Código de ejemplo de una estructura try/catch/finally

Dividir entre 0 no produce un error en JavaScript, solamente genera un valor *infinity*. Con el siguiente código vamos a utilizar la sentencia *throw* para lanzar un error cuando el denominador sea 0 con el mensaje “el denominador no puede ser 0”.

El siguiente ejemplo mostrará dos alertas con los mensajes:

Error: "el denominador no puede ser 0"

Cuidado con no declarar las variables

```
let x=20, y=0,r;
try {
   if(y===0){
      throw "el denominador no puede ser 0";
   }
   r= x/y;
   alert(r);  //ésta alerta nunca se lanza
}
catch (err) {
  alert("Error: " + err);

} finally {
  alert("Cuidado con no declarar las variables") ;
}
```

Código de ejemplo en el que se lanza una excepción personalizada con throw

1.2.11. Mostrar mensajes

En el transcurso del módulo veremos distintas estrategias para interaccionar con el usuario pidiendo y mostrando información. Vamos a ver a continuación las formas más básicas de mostrar información

1.2.11.1. Mostrar un mensaje con una ventana emergente

La forma más básica para mostrar un mensaje al usuario es lanzando una ventana emergente mediante la sintaxis:

```
window.alert("mensaje a mostrar al usuario");
```

Cuando se ejecute esta sentencia se mostrará una ventana emergente al usuario el mensaje escrito entre los paréntesis y un botón para aceptar y cerrar la ventana.

Esta forma de mostrar datos es la menos recomendable por ser muy molesta y poco usable. Además, el *alert* detiene por completo la ejecución del código JavaScript y la carga de la web hasta que el usuario no acepta.

1.2.11.2. Mostrar un mensaje por consola

Si queremos mostrar mensajes de depuración que no sean visibles a los usuarios en general, podemos mostrar mensajes en la consola del navegador con la sintáxis:

```
console.log("mensaje a mostrar por consola");
```

Esta forma es la más recomendable a utilizar mientras estamos programando y queramos mostrar información de forma rápida para conocer el estado de nuestro programa. Pero hay que tener presente que éstos mensajes solo se mostraran si el navegador tiene abiertas las herramientas de desarrollo y depuración, por lo que son mensajes que el usuario final seguramente no verá (y en general deberían eliminarse en producción).

1.2.11.3. Mostrar un mensaje en el HTML

Esta es la forma más compleja y más recomendable de mostrar un mensaje al usuario. Consiste en modificar el contenido de un elemento HTML con el mensaje que queramos que tenga. Para poder acceder a un elemento HTML desde JavaScript deberemos seguir dos pasos:

- Asegurarnos que el elemento HTML ya se ha cargado para que podamos acceder a él. Para no tener errores de momento nos conformaremos con escribir el script después de declarar el elemento.
- Conocer una manera de acceder a ése elemento en concreto y modificar su contenido. De momento, nosotros accederemos a él a partir de su atributo id y modificaremos su contenido con la sentencia: document.getElementById("idDelElemento").innerHTML="nuevo valor";

Vamos a ver entonces un código de ejemplo que modificará el contenido de un DIV con el texto inicial “HOLA” para que pase a tener el texto “ADIOS”

```
<div id="mensaje"> HOLA </div>
<script>
    document.getElementById("mensaje")="ADIOS";
</script>
```

1.2.12. Introducción a los eventos

Una de las características principales de JavaScript es su capacidad para ejecutar funciones cuando se produce algún tipo de interacción con el documento. Normalmente estas interacciones suelen ser producidas por el usuario (al clicar o teclear), pero también las puede producir el mismo navegador o JavaScipt.

Este tipo de interacciones se llaman *Eventos*, y vamos a ver a continuación los dos principales eventos con los que trabajaremos en JavaScript. El evento onclick y el evento onload.

1.2.12.1. Onclick

Este evento se produce cuando el usuario clica sobre un elemento. Con JavaScript podemos vincular éste evento a cualquier elemento del HTML de distintas formas que veremos más a delante, pero por ahora veremos la más básica: añadiendo el tributo "onclick" a aquel elemento HTML sobre el que queramos vincular el evento y asignándole el código JavaScript que queramos ejecutar.

En el siguiente ejemplo cuando se clique encima el div con el texto PRESS se realizará la operación matemática y mostrará el resultado por consola.

```
<div onclick="let resul=5+6; console.log(resul);"> PRESS </div>
```

Pero no es una buena idea escribir el código directamente en el HTML. Para evitarlo podemos crear una función en JavaScript y al clicar encima PRESS ejecutar dicha función tal y como hemos programado en el siguiente ejemplo:

```
<head>
    <script>
    function sumar(){
        let resul = 5+6;
        console.log(resul);
    }
   </script>
</head>
<body>
    <div onclick="sumar();"> PRESS </div>
</body>
```

1.2.12.2. Onload

Este evento se produce cuando el navegador ha terminado de cargar y renderizar todo el contenido del documento HTML. A diferencia del evento *onclick*¸*el evento onload* solo se puede añadir al elemento BODY.

En el siguiente ejemplo se mostrará por consola la operación matemática justo después de haber cargado la página web.

```
<head>
     <script>
    function sumar(){
        let resul = 5+6;
        console.log(resul);
    }
   </script>
</head>
<body onload="sumar()">

</body>
```

1.2.13. Pedir datos al usuario

En el transcurso del módulo veremos distintas estrategias para interaccionar con el usuario. Vamos a ver a continuación las formas más básicas de pedir información:

1.2.13.1. Pedir datos con una ventana emergente

La forma más básica para pedir un dato al usuario es utilizando una ventana emergente mediante la sintaxis:

```
let respuesta = window.prompt("mensaje a mostrar al usuario");
```

Cuando se ejecute esta sentencia se mostrará una ventana emergente al usuario con una caja de texto en la que el usuario va a poder escribir algo y un botón de “ACEPTAR” y “CANCELAR”.

En el caso que el usuario clique en ”ACEPTAR” la variable *respuesta* obtendrá el texto que haya escrito el usuario. Si el usuario escribe un número, igualmente *respuesta obtendrá un texto con el número (por ejemplo “6” o “8”)*. Si el usuario no escribe nada *respuesta* contendrá una cadena vacía.

En el caso que el usuario clique en “CANCELAR” la variable *respuesta* obtendrá el valor *null*.

Esta forma de pedir datos es la menos recomendable por ser muy molesta y poco usable. Además, el *promt* detiene por completo la ejecución del código JavaScript hasta que el usuario no escoge una opción.

1.2.13.2. Pedir datos a través de un input

Una forma más amigable de pedir datos al usuario es utilizando los inputs del HTML. Cuando trabajemos con formularios veremos con más profundidad como acceder a los *inputs*, sus tributos y su contenido, pero por ahora veremos una forma básica de obtener la información que el usuario escriba en un input de tipo texto a partir de su atributo *id* con la sentencia: *let texto = document.getElementById("idInput").value;*

En el siguiente ejemplo utilizamos los conocimientos de eventos que hemos visto para que al clicar sobre el botón LEER se muestre en el div con id "resultado" el texto escrito en el input con id "entrada".

```
<head>
    <script>
    function leerInput(){
        let texto = document.getElementById("entrada").value
        document.getElementById("resultado").innerHTML=texto;
    }
    </script>
</head>
<body>
   <input type="text" id="entrada">
   <button onclick="leerInput();">LEER</button>
    <div id="resultado"></div>
</body>
```

Código de ejemplo para mostrar texto en un elemento HTML al clicar sobre un botón.

En el siguiente vídeo veremos las distintas peticiones que se hacen al servidor, el proceso de carga de una página web, el proceso de ejecución y la depuración en la entrada y salida de datos.

Video: Carga y entrada y salida de datos en JavaScript

Recursos y enlaces

- Sitio oficial de Visual Studio Code

- Sitio oficial de ASP.NET

- Sitio oficial de Apache Server

- Sitio oficial de Nginx

- Sitio oficial de IIS

- Sitio oficial de NodeJS

Conceptos clave

- **Web dinámica:** aquella cuyo contenido varía según el momento o los parámetros enviados.
- **Cliente web:** comúnmente un "navegador web", es un software que envía peticiones al servidor e interpreta las respuestas mostrándolas al usuario.
- **Servidor web**: software encargado de recibir consultas de un cliente, evaluarlas y retornar lo que el cliente le pide.
- **Ámbito de una variable**: el bloque de código en el que esa variable está disponible.

Test de autoevaluación

¿Cuál de las siguientes tecnologías es interpretada por un cliente web .

a) PHP
b) ASP
c) JavaScript
d) JSP

¿En una arquitectura de 4 capas, la 3ª capa con qué otras capas puede interactuar?

a) Solo con la 2a
b) Solo con la 4a
c) Con todas las capas
d) Solo con la 2ª y la 4ª

¿Qué método es el más amigable para pedir datos al usuario desde JavaScript?

a) Con un alert
b) Por la consola
c) Con un input
d) Desde JavaScript no se pueden pedir datos al usuario

¿Qué operador compara si dos valores són iguales en tipo y valor?

a) ===
b) ==
c) !==
d) <=>

Ponlo en práctica

Actividad 1

Crea un documento HTML y vincúlale dos archivos JS externos y dos imágenes.

Utilizando el inspector de res de las herramientas de desarrollo del navegador, mira cuantas peticiones se realizan para cargar la web

Crea un diagrama de comunicación para mostrar las peticiones realizadas necesarias para cargar todo el contenido web.

Actividad 2

Crea un documento HTML con un INPUT, un botón y dos DIV. Cuando el usuario clique en el primer botón pide un número con un *prompt* y muestre si es par o impar con un *alert*. Cuando el usuario clique en el segundo botón obtén el valor escrito en el INPUT y muestra si es par o impar en el DIV.

SOLUCIONARIOS

Test de autoevaluación

¿Cuál de las siguientes tecnologías es interpretada por un cliente web .

e) PHP
f) ASP
g) JavaScript
h) JSP

¿En una arquitectura de 4 capas, la 3ª capa con qué otras capas puede interactuar?

e) Solo con la 2a
f) Solo con la 4a
g) Con todas las capas
h) Solo con la 2ª y la 4ª

¿Qué método es el más amigable para pedir datos al usuario desde JavaScript?

e) Con un alert
f) Por la consola
g) Con un input
h) Desde JavaScript no se pueden pedir datos al usuario

¿Qué operador compara si dos valores són iguales en tipo y valor?

e) ===
f) ==
g) !==
h) <=>

Ponlo en práctica

Actividad 1

Crea un documento HTML y vincúlale dos archivos JS externos y dos imágenes.

Utilizando el inspector de res de las herramientas de desarrollo del navegador, mira cuantas peticiones se realizan para cargar la web

Crea un diagrama de comunicación para mostrar las peticiones realizadas necesarias para cargar todo el contenido web.

Los solucionarios están disponibles en la versión interactiva del aula.

Actividad 2

Crea un documento HTML con un INPUT, un botón y dos DIV. Cuando el usuario clique en el primer botón pide un número con un *prompt* y muestre si es par o impar con un *alert*. Cuando el usuario clique en el segundo botón obtén el valor escrito en el INPUT y muestra si es par o impar en el DIV.

Los solucionarios están disponibles en la versión interactiva del aula.