HTML5

* Tags, Best Practice y Validators -

**Turno: Mañana**

**ADA ITW**

**- Grupo N°6 -**

**Integrantes:**

* Aponte, Elifer.
* Beccan, Constanza.
* Busatto, María Florencia.
* Garcete, Erica.
* Yusti, Zoe.

**Introducción:**

El presente trabajo pretende afianzar los conocimientos sobre HTML, obtenidos durante la cursada en ADA ITW, poniendo especial énfasis en las nuevas etiquetas incorporadas en la 5ta versión y sin dejar de lado el apartado de las buenas prácticas que debe observar todo *developer* que se precie de tal y el uso correcto de validators que nos permita trabajar con un código “limpio y depurado de errores”.

A lo largo de los distintos ejes temáticos de nuestro trabajo, el apartado de FAQ y nuestra presentación dinámica esperamos ayudar a reforzar conceptos de una forma amena y respetando un vocabulario técnico que colabore en la incorporación de dichos conceptos a nuestra tarea diaria.

¡Esperamos que lo disfruten!

**Las autoras.**

**¿Qué es HTML?**

Haciendoreferencia a sus siglas en inglés***, el HyperText Markup Language*** (lenguaje de marcas de hipertexto) es una forma de embeber texto plano. Este nació en 1980 a partir de una propuesta de un nuevo sistema de *"hipertexto"* para compartir documentos. Antiguamente, este sistema se utilizaba en el ámbito informático para la visualización de documentos electrónicos. De cierta manera, los primitivos sistemas de *"hipertexto"* podrían asimilarse a los enlaces de las páginas web actuales.

El primer documento formal con la descripción de HTML se publicó en 1991 bajo el nombre***HTML Tags*** *(Etiquetas HTML*) y todavía hoy puede ser consultado online a modo de *reliquia informática* en el siguiente link : [**https://tinyurl.com/oldhtmltags**](https://tinyurl.com/oldhtmltags)

**¿Cómo funciona el HTML?**

Todo HTML se basa en un único concepto, las etiquetas. Estas son una forma de hablarle al navegador. Al crear un archivo con etiquetas e información, básicamente le estamos dando instrucciones al navegador de como mostrar dicha información. Con esto, si creamos una etiqueta diciendo “esto es un título” el navegador interpretará la información dentro de la etiqueta y mostrará la información en forma de título.

**Etiquetas agregadas para HTML5**

**Secciones:**

|  |  |
| --- | --- |
| [<section>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/section) | Define una sección en un documento. |
| [<nav>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/nav) | Define una sección que solamente contiene enlaces de navegación. |
| [<article>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/article) | Define contenido autónomo que podría existir independientemente del resto del contenido. |
| [<aside>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/aside) | Define algunos contenidos vagamente relacionados con el resto del contenido de la página. Si es removido, el contenido restante seguirá teniendo sentido. |

|  |  |
| --- | --- |
| [<header>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/header) | Define la cabecera de una página o sección. Usualmente contiene un logotipo, el título del sitio Web y una tabla de navegación de contenidos. |
| [<footer>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/footer) | Define el pie de una página o sección. Usualmente contiene un mensaje de derechos de autoría, algunos enlaces a información legal o direcciones para dar información de retroalimentación. |
| [<address>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/address) | Define una sección que contiene información de contacto. |
| [<main>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/main) | Define el contenido principal o importante en el documento. Solamente existe un elemento <main> en el documento. |

**Agrupación de contenido:**

|  |  |
| --- | --- |
| [<figure>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/figure) | Representa una figura ilustrada como parte del documento. |
| [<figcaption>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/figcaption) | Representa la leyenda de una figura. |

**Semántica a nivel de texto:**

|  |  |
| --- | --- |
| [<data>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/data) | Asocia un *equivalente legible por máquina* a sus contenidos. (Este elemento está sólamente en la versión de la WHATWG del estándar HTML, y no en la versión de la W3C de HTML5). |
| [<time>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/time) | Representa un valor de *fecha* y *hora*; el equivalente legible por máquina puede ser representado en el atributo datetime. |

|  |  |
| --- | --- |
| [<mark>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/mark) | Representa texto resaltado con propósitos de *referencia*, es decir por su relevancia en otro contexto. |
| [<ruby>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/ruby) | Representa contenidos a ser marcados con *anotaciones ruby*, recorridos cortos de texto presentados junto al texto. Estos son utilizados con regularidad en conjunto a lenguajes de Asia del Este, donde las anotaciones actúan como una guía para la pronunciación, como el *furigana* Japonés. |
| [<rt>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/rt) | Representa el *texto de una anotación ruby* . |
| [<rp>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/rp) | Representa los *paréntesis* alrededor de una anotación ruby, usada para mostrar la anotación de manera alterna por los navegadores que no soporten despliegue estandar para las anotaciones. |
| [<bdi>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/bdi) | Representa un texto que debe ser *aislado* de sus alrededores para el formateado bidireccional del texto. Permite incrustar un fragmento de texto con una direccionalidad diferente o desconocida. |

|  |  |
| --- | --- |
| [<wbr>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/wbr) | Representa una *oportunidad de salto de línea,* es decir, un punto sugerido de envoltura donde el texto de múltiples líneas puede ser dividido para mejorar su legibilidad. |

**Contenido Incrustado:**

|  |  |
| --- | --- |
| [<embed>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/embed) | Representa un *punto de integración* para una aplicación o contenido interactivo externo que por lo general no es HTML. |

|  |  |
| --- | --- |
| [<svg>](https://developer.mozilla.org/es/docs/Web/SVG/Element/svg) | Define una *imagen vectorial* embebida. |
| [<math>](https://developer.mozilla.org/es/docs/Web/MathML/Elemento/math) | Define una *fórmula matemática*. |

Formularios:

|  |  |
| --- | --- |
| [<datalist>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/datalist) | Representa un *conjunto de opciones predefinidas* para otros controles. |

|  |  |
| --- | --- |
| [<keygen>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/keygen) | Representa un control de *par generador de llaves.* |
| [<output>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/output) | Representa el *resultado de un cálculo.* |
| [<progress>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/progress) | Representa el *progreso de finalización* de una tarea. |
| [<meter>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/meter) | Representa la *medida* escalar (o el valor fraccionario) dentro de un rango conocido. |

**Elementos Interactivos :**

|  |  |
| --- | --- |
| [<details>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/details) | Representa un *widget* desde el que un usuario puede obtener información o controles adicionales. |
| [<summary>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/summary) | Representa un *resumen, título* o *leyenda* para un elemento <details> dado. |
| [<command>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/command) | Representa un *comando* que un usuario puede invocar. |
| [<menu>](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/menu) | Representa una *lista de comandos* . |

**¿De qué hablamos cuando nos referimos a Meta Tags?**

Las **Meta Etiquetas** o **Meta Tags** son etiquetas de información que se añaden en el código html de cada página de una Web para aportar información relevante sobre la categorización de esa página web. Es una información que sólo podrá accederse a través del código fuente y que en realidad sólo resulta relevante para los buscadores, pero su finalidad es de gran utilidad ya que aportan gran información sobre la página como el autor, fecha, palabras clave, descripción, etc.

La calidad de una **meta tag** tiene dos aspectos importantes:

1. Como una persona evalúa la **meta tag**
2. Como un robot evalúa la **meta tag.**

Para una persona, la **meta tag** necesita ser llamativa, interesante, informativa, curiosa y con un toque de “*call-for-action”* (instrucción que provoca una reacción inmediata).

Para un robot, la **meta tag** necesita de una dosis de palabras clave para que el algoritmo clasifique de que trata la página web.

Lo esencial en el desarrollo de la **meta tags** es encontrar un término medio entre estos dos puntos.

**La Meta Etiqueta Title:** Estrictamente hablando, este no es ningún meta-tag, sino una etiqueta autónoma de HTML, aunque, debido a su significado a la hora de interactuar con los agentes de usuario es común que sea mencionada como parte de los metadatos. Actúa como título de la página en cuestión y debe estar formada por palabras clave presentes en la página web.

**Codificación de caracteres:** Si la fuente no fue previamente definida en el header del archivo HTTP, es necesario hacerlo usando HTML. Así se evita, por ejemplo, que la ñ o las tildes no se muestren correctamente. **La Meta Etiqueta Description:** La etiqueta descripción o Meta Etiqueta Description actúa como una descripción de la página en cuestión y también debe estar formada por palabras clave y frases que resuman el contenido de la página web. Esta información se muestra como snippet (una síntesis en dos líneas del tema de una página que aparece bajo la URL) en los buscadores de uso más generalizado como Google o Bing, por lo que se recomienda cuidar su redacción. **Palabras clave (Keywords):** Con esta etiqueta meta los administradores tienen la posibilidad de definir palabras clave para el buscador. Las keywords son aquellos criterios a los que responde un buscador para ofrecerle al usuario páginas HTML como respuesta, donde tales palabras clave son parte de los meta tags. Autor (author) y copyright: Estos dos meta tags, de uso opcional desde el punto legal, permiten hacer referencia al diseñador de una página web y al propietario de los derechos del código fuente de una página HTML.

**VIEWPORT**

Básicamente, sirve para definir qué área de pantalla está disponible al renderizar un documento, cuál es el nivel de escalado que puede realizar el usuario, así como si el navegador debe mostrarla con algún *zoom* inicial. Todo ello se indica a través de varios parámetros en la propia etiqueta META.

**Lista completa de propiedades de viewport:**

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Valores** | **Descripción** |
| **width** | Valor integral (en pixeles)  o constante device-width | Define el ancho del viewport. |
| **height** | Valor integral (en píxeles)  o constante device-height | Define el alto del viewport. |
| **initial-scale** | Cualquier número real de 0.1 en adelante. 1 representa no escala. | Escala inicial del viewport. |
| **minimum-scale** | Cualquier número real de 0.1  en adelante. 1 representa no escala. | Escala máxima del viewport. |
| **maximum-scale** | Cualquier número real de 0.1 en adelante. 1 representa no escala. | Escala mínima del viewport. |
| **user-scale** | "yes" / "no" | Permiso para que el usuario pueda hacer zoom. |

**¿Cuales son las buenas prácticas en un Programador?**

* **Nunca perder el foco del proyecto:** para esto podemos hacer un documento, un diagrama de flujo, mock ups y otras herramientas para esquematizar, para que durante el desarrollo no perdamos el objetivo principal del mismo.
* **Usar comentarios:**todo programa debe ser previamente comentado, explicando el propósito, funcionamiento completo y resultado esperado. Adicionalmente a los comentarios generales, siempre comenta antes de cada conjunto importante de código. **¿Qué evitar?** el código ***commented-out*,** que corresponde al código comentado para que no se ejecute/no compile, ya que la lectura del código se vuelve engorrosa.
* **Keep it simple:**escribe tus programas lo más simple y directo posible.
* **Uso de compiladores:** si trabajas con un compilador, ajusta sus opciones para que arroje la mayor cantidad de errores y advertencias posibles, así existirán menores chances de errores aleatorios.
* **Identación:** dentro de las funciones definidas, establece un **espaciado**  que resalte la estructura funcional y facilite la lectura al programador al que le corresponda analizar el código:

1. **Por lo general, se usa un nivel de indentación por cada bloque de código**, ésta indentación corresponde a una sangría que comúnmente tiene el valor de una tabulación (tecla Tab) o bien tres o cuatro espacios.
2. **Es importante que el tamaño de las sangrías sean regulares (consistentes) y no varíen a lo largo del código**.

* **Variables:** se recomienda declarar variables en líneas separadas, ya que se facilita la descripción de cada variable mediante comentarios:

1. **No uses variables cuyo nombre no posea significado descriptivo**, una variable con nombres significativos permite al lector entender el contexto del código y permite disminuir la cantidad de documentación asociada, puesto que con un código legible y nombres significativos, el código se ve *auto documentado*.
2. **Definir l**as variables locales al inicio de la implementación de cada función, como un bloque de código bien separado del bloque que contenga las instrucciones ejecutables, ésta separación puede consistir en una línea en blanco, o un comentario que denota la utilidad de cada bloque.

* **Sobre las comas:** poner un espacio después de cada coma(,) facilita la legibilidad del código.
* **Elección de nombres:** se consistente al usar un estándar para nombres largos, puedes usar el estándar usado en C ("nombre\_largo"), o el de Java, llamado CamelCase("nombre","VariableNombreLargo","Clase","ClaseNombreLargo").
* **Comentarios en las Funciones:** comenta cuando sea justo y necesario, usa los comentarios dentro de las funciones para describir las variables y cuando existan bloques de código difíciles de entender el exceso de comentarios vuelve ilegible el código.
* **Operadores:**

1. En caso de usar operadores binarios (por ejemplo +, -, &&, ||, entre otros) se recomienda poner espacio a los extremos de cada operador, de modo que se resalte su presencia y se facilite la lectura del código.
2. Se recomienda en algunas operaciones complejas, hacer uso de paréntesis redundantes o innecesarios que sirven para poder agrupar expresiones dentro de tales operaciones.

* **Evita la incorporación de más de una instrucción por línea:**  esto reduce la legibilidad del código, ya que el programador habitualmente está acostumbrado a leer una instrucción por línea.
* **Uso de llaves:** si el lenguaje soporta llaves({}) para la separación de bloques, es recomendable usarlas, esto facilita la distinción de bloques de código en forma rápida, permitiendo identificar y reparar errores con menor dificultad.
* **Contadores: nunca** olvides inicializar los contadores y sumadores
* **Usar minúsculas:** no solo en las etiquetas, sino también en los atributos para facilitar la lectura de tu código.

**Buenas prácticas en HTML5**

**Además de las reglas generales, veamos algunas prácticas específicas de HTML5 que nos ayudarán a la hora de un correcto maquetado.**

1. **Hacer uso del Doctype**: la declaración de un “Doctype” se realiza al inicio del código. **En la declaración NO debe haber NINGÚN espacio en blanco ni cambio de línea ANTES**, la mayoría de los navegadores saben arreglárselas, pero en IE provocará “misteriosos comportamientos”.
2. **Uso preciso de las etiquetas:** usar de la forma más precisa posible las etiquetas de HTML5 en lugar de abusar del uso del <div>. A modo de ejemplo, lo correcto sería utilizar: **<footer> En lugar de: <div class="footer">**
3. **CSS Reset:** es válido usar un **CSS Reset** para resetear desde cero ciertos estilos que por default trae algunos elementos, también puede usarse Normalize.css.
4. **Enlace de archivos en HTML:** enlazar **los estilos CSS** que vamos a utilizar al inicio de nuestro HTML. Los archivos externos JavaScript deben enlazarse al final de nuestro HTML antes del cierre de la etiqueta </body>
5. **Usar los atributos “alt” y “title” para las imágenes:** el atributo **“alt”** significa “texto alternativo”, y quiere decir, que si no se encuentra la imagen se debe mostrar ese texto, entonces un error común es introducir toda una oración, lo cual rompería la página si le pedimos al navegador que no muestre las imágenes. Por otra parte los buscadores comprenden el significado de la imagen por su **“title”,** el título sí puede llevar una oración explicativa.
6. **Encoding** **UTF-8 :** Especificar el meta para el encoding, de modo que no haya ninguna necesidad de utilizar HTMLentities para las tildes y caracteres raros.
7. **NO utilizar tablas, A MENOS que se trate de información tabulada:** si la naturaleza de la información es una relación de datos tabulados, entonces nada mejor que una tabla para representarla.
8. **Separa el contenido de la página HTML del estilo con que se muestra:** Siempre hay que usar estilos separados y no dentro de la página HTML, así el código será más limpio y permitirá modificaciones de manera más eficaz.
9. **Unificar todos los estilos en una sola hoja de estilos:** cada archivo implica una solicitud HTTP, lo cual hace que el tiempo de carga de la página sea más lento. Por eso es mejor usar un framework que permite “agregar” los ficheros CSS y JS en un único fichero para CSS y uno para JS.
10. **No abusar de float:** esto es muy importante, para garantizar la compatibilidad entre los diferentes navegadores, **los divs no deben ser flotados masivamente** porque en muchos browsers (IE por ejemplo) el sitio se rompe de mala manera.

Estructurar un proyecto en Programación

Convertirse en programador no solo es aprender la sintaxis y los conceptos de un determinado lenguaje: se trata de emplear ese conocimiento para hacer programas. Por eso hablaremos del proceso en general y en particular para realizar el desarrollo.

Pautas Generales

**1 - Seleccionar el talento y los recursos apropiados:** realizar un proceso de selección del talentocon las destrezas necesarias y experiencia relevante es vital para garantizar el éxito del proyecto. Tampoco hay que escatimar al invertir en herramientas que aumentan la productividad y eficiencia del equipo de desarrollo: hardware moderno, software y plataformas de desarrollo y de pruebas actualizado, etc. Esto ayuda a que el equipo pueda imprimir todo su conocimiento para garantizar un producto sólido, fiable y robusto.

**2 - Escoger el proceso de desarrollo apropiado:** el ciclo de vida del desarrollo del software tiene una fuerte dependencia del proceso elegido. El modelo en cascada, la metodología ágil, el enfoque iterativo en espiral, son todas formas contrastadas de alcanzar el éxito. La dificultad está en elegir bien qué metodología le conviene más a cada tipo de proyecto.

**3 - Presupuestos y estimaciones razonables:** muchos proyectos fracasan o se prolongan en el tiempo por hacer estimaciones poco realistas. Una planificación razonable depende de [fijar bien los tiempos, el presupuesto, los recursos y los esfuerzos](https://velneo.es/cuanto-vale-tu-software/).

**4 - Fijar hitos más pequeños:** los grandes proyectos deben complementarse con mini-hitos para poder hacer mejor seguimiento, mejor gestión de riesgos, y para mitigar incidencias de forma controlada.

**5 - Definir bien los requisitos:** documentar de forma efectiva los requisitos es la columna vertebral para poder alinear el producto final con los objetivos empresariales. Es imperativo que se reúnan todas las partes (clientes, responsables de empresa y los líderes de los equipos) para documentar los requisitos de forma clara, sin dejar lugar a dudas o a la improvisación.

**6 - Implementar el código de manera efectiva:** el uso de módulos más pequeños que están auto-probados, probados unitariamente y que se integran continuamente es una buena práctica muy extendida, se recomienda para garantizar que la funcionalidad ya implementada no rompe.

**7 - Pruebas rigurosas y validación:** la planificación de pruebas, la creación de conjuntos de pruebas y la ejecución de las mismas son muy importantes con el fin de [validar la funcionalidad desarrollada](https://velneo.es/estabilidad-en-el-software/). De hecho, la planificación de las pruebas debe hacerse en paralelo a la fase de desarrollo. Igual de importante es la documentación que hagamos de las pruebas, informar de forma efectiva los errores, el rastreo de los errores y la corrección de los mismos.

**8 - Documentación:** al igual que el propio software, es importante toda la documentación sobre el que se apoya . Estos documentos ayudan a garantizar el entendimiento del software, trazabilidad y eliminar la dependencia del equipo de desarrollo original. Pueden usarse como referencia en el futuro para mantener, mejorar o usar el software.

**9 - Planificar sesiones de revisión de código:** las revisiones de código muchas veces son más efectivas para encontrar errores que si solo hacemos pruebas. Las revisiones de todos los entregables, del código y de la documentación es algo que siempre se debe hacer.

**10 - Garantizar la gestión del control de las fuentes del software:** el uso de una gestión efectiva del código fuente y la documentación, para que estén controlados según la versión, es fundamental para mantener la trazabilidad y la reversión controlada del código si fuera necesario.

**11 - Control de calidad:** ayuda a sacar adelante los proyectos de desarrollo sin graves trastornos y de forma más rápida. Desde la detección de fallos hasta el establecimiento de métricas claves, las mejores prácticas en este terreno han demostrado ser un éxito a la hora de determinar si un proyecto está en condiciones de pasar a una nueva fase, si está listo para ser lanzado o entregado al cliente.

**12 - Instalación y despliegue eficaz:** en muchas ocasiones cuando ya hemos probado el software y todo va bien, de repente el proyecto fracasa en casa del cliente o en fase de implementación y despliegue. Es muy importante tener un buen plan de despliegue y hacer una lista a modo de “checklist” para evitar desastres.

**13 - Soporte y mantenimiento:** Incluso cuando el software está funcionando, debe haber un proceso de soporte y mantenimiento, previamente diseñado y pensado, para poder informar y reportar errores y mejoras al equipo de desarrollo de forma eficaz.

Pautas específicas:

Al iniciar nuestro desarrollo podemos recurrir a distintos modelos para esquematizar las etapas de nuestro proyecto. Los más conocidos son : **el modelo de desarrollo en cascada y el modelo de desarrollo en espiral.**

Por su sencillez explicamos brevemente cómo estructurar los pasos de nuestro desarrollo bajo el modelo de cascada, el cual **define que las siguientes fases deben cumplirse de forma sucesiva:**

1. **Especificación de requisitos.**
2. **Diseño del software.**
3. **Construcción o Implementación del software.**
4. **Integración.**
5. **Pruebas (o validación)**
6. **Despliegue (o instalación)**
7. **Mantenimiento.**

Código redundante: ¿Como evitarlo?

¿Qué es el código redundante? En programación, se conoce como código redundante a cualquier parte del código fuente que tenga algún tipo de redundancia tales como recalcular un valor que ha sido calculado previamente y todavía está disponible.

Para eliminar el código redundante, uno de los principios más difundidos es el llamado DRY(don't repeat yourself), que hace alusión básicamente a evitar que el mismo código exista en distintas partes. Se busca lograr a fin de cuentas un código elegante: legible, óptimo en cuanto a rendimiento, simple y sin líneas redundantes o innecesarias.

# **Estándares de Programación**

# 

# **¿Cuál es el estilo adecuado?**: a decir verdad, no existe un "estilo" mejor que otro, pero para sortear el inconveniente de no saber qué estilo elegir suelen seguirse los denominados **"estándares de programación"**.

# Un estándar de programación es: **“una forma de *normalizar* la programación de forma tal que al trabajar en un proyecto, cualquier persona involucrada en el mismo tenga acceso y comprenda el código”.**Esto nos permite:

# Definir la escritura y organización del código fuente de un programa.

# Facilita a otro programador la modificación de tu propio código fuente aunque no estés trabajando en el equipo.

# Definir la forma en que deben ser declaradas las variables, las clases, los comentarios.

# Especificar qué datos deben incluirse acerca del programador y de los cambios realizados al código, etc.

# **Nombres de variables y programas**

## **Variables**

* Los nombres que se usen deben ser significativos.
* Los nombres deben estar en minúsculas, excepto la primera letra de cada palabra a partir de la segunda.
* Una variable $aa o $a1 no significan nada. No hay problema en usarlo si es una variable temporal que va a ser empleada en las líneas siguientes, pero si va a ser utilizada más lejos en el programa, debe tener un nombre significativo.

**Constantes y Variables globales**

**Constantes:**

* Se deben evitar constantes numéricas sin mucho significado,también es conveniente definir las constantes en el programa. Todos los caracteres deben estar en mayúsculas y las palabras separadas por "\_".

# **Corchetes e indentación**

* La indentación ayuda a dar claridad a un programa y es necesario que se haga bien. Debe hacerse con "tabs" y no con espacios en blanco. **Los corchetes de un bloque if, o switch, o for, deben ir en la misma línea de la cláusula.**

**Claridad de los programas**

* Los programas y rutinas que se escriban deben ser fáciles de entender. Por eso, además de dar la explicación de que hace cada programa o función al principio, hay que tratar que las funciones quepan en una sóla página y que antes de cada sección se explique qué es lo que se está haciendo.

**Inclusión de funciones y rutinas**

* Cuando se incluye un archivo que tiene muchas funciones, es muy importante que al hacer el **require** del archivo, que se indiquen los nombres de las funciones que se están utilizando. De forma que cuando se quiera saber de donde viene una función se pueda, al buscar la primera ocurrencia del nombre.

**Ofuscación de código CSS**

**¿Cómo funciona?**

Es una forma única de proteger sus hojas de estilo, transformandolas de una manera que nadie querrá modificarlas. Un ofuscador también se encarga de minimizar y hacer modificaciones al programa, fusionando los nombres de variables, funciones y miembros.

Se pueden encontrar dos tipos de ofuscación de código. La ofuscación superficial siendo solamente un reacomodo de la sintaxis del programa y la ofuscación profunda que intenta cambiar la estructura actual del programa, cambiando su control de flujo o modo de referenciar sus datos.

**Ejemplo de un código sin ofuscar**

**var a="Hello World!";**

**function MsgBox(msg)**

**{**

**alert(msg+"\n"+a);**

**}**

**MsgBox("OK");**

**Código ofuscado**

**var \_0x55ae=["\x48\x65\x6C\x6C\x6F\x20\x57\x6F\x72\x6C\x64\x21","\x0A","\x4F\x4B"];var a=\_0x55ae[0];function MsgBox(\_0x9e43x3){alert(\_0x9e43x3+\_0x55ae[1]+a);} ;MsgBox(\_0x55ae[2]);**

**Ventajas y desventajas de ofuscar código**

**Ventajas**

* Evita que tu código fuente sea obtenido por alguien más.
* Genera múltiples versiones del mismo código.
* Enmascara el código para evitar que se pueda determinar la forma que tenía originalmente.

**Desventajas**

* La ofuscación del código puede ser más difícil de reconstruir.
* Si es utilizada como una arma para lograr burlar los actuales mecanismos de defensa puede ser muy peligrosa.
* El mantenimiento y la solución de problemas de una aplicación son tareas más difíciles.

**HTML semántico**

Usar HTML semántico significa aplicar a cada parte del contenido la etiqueta más adecuada a su tipo, por ejemplo: para los párrafos se usa [<p>](http://www.w3schools.com/TAgs/tag_p.asp) y para las listas, [<ol>](http://www.w3schools.com/TAgs/tag_ol.asp) o [<ul>](http://www.w3schools.com/TAgs/tag_ul.asp).

Utilizar código semántico ofrece varias ventajas:

**Posicionamiento en buscadores**. Los motores de búsqueda analizan el código para saber qué clase de contenido muestran y cómo deberían ser mostrados.

**Accesibilidad**. Los lectores de pantalla para usuarios ciegos organizan la lectura del contenido de acuerdo con la estructura del código.

**Practicidad**. El código semántico es más fácil de entender y mantener.

**Reusabilidad**. La separación del contenido de la presentación permite que una página sea rediseñada cambiando sólo el CSS.

A medida que los desarrolladores cumplan con mejorar el significado de su código avanzaremos hacia la [Web Semántica](http://www.4rsoluciones.com/blog/que-es-la-web-semantica-2/). Teniendo como resultado un contenido bien estructurado para los buscadores, donde el usuario podrá encontrar exactamente lo que desea sin estar navegando de página en página.

**Validator**

**¿Qué es un validador?**

Se trata de servicios normalmente gratuitos que evalúan el código (tanto del contenido HTML como de la presentación CSS), e informan si tienen errores o están bien escritos. Cumplen una función similar al corrector de ortografía y gramática de Microsoft Word. Pero no solo se encargan que no tengan errores,si no que algunos te dan el código ya limpio.

Los profesionales web eligen validar porque es una muy buena herramienta de debugging, los browsers actualmente hacen un buen trabajo resolviendo hasta la peor sopa de etiquetas pero algunos errores se escapan con gracia. Muy a menudo diferentes programas en diferentes plataformas no manejan los errores de manera similar.

De hecho, la mayoría de los desarrolladores que crean aplicaciones web saben que su código necesita ser analizado, asegurando así, que su código, las etiquetas y el CSS estén validados antes de crear la capa interactiva.

A raíz de una encuesta, la gran mayoría de los profesionales de la Web afirman que los errores de validación son lo primero que verificarán cada vez que se topan con un error de estilo o creación de scripts en la web.

Es muy popular el dicho que ronda “Google es ciego y gusta de las webs bien estructuradas.”,en otras palabras, si tu sitio es válido, tus SEO aumentarán.

**¿Para qué validar?**

La validación es como un control de calidad a prueba de futuros, comprobar que una página "se muestra bien" en algunos buscadores puede ser un signo de que la página "funcionará" hoy, pero no garantiza que funcionará mañana.

La validación es una de las maneras más sencillas de comprobar si una página está construida de acuerdo con los estándares web, y proporciona una de las garantías más confiables de que las plataforma seguirá existiendo en el futuro.

La validación facilita el mantenimiento, esto es crucial para seguir con el proyecto a pesar de si lo continua 1 o más personas.

La validación nos enseña buenas prácticas, al adquirir esta norma como rutina los devs forman buenas prácticas de trabajo.

Se hace hincapié en que las pruebas de validación automatizadas son una buena introducción a conceptos de calidad más amplios y más complejos, como la accesibilidad.

La validación es un signo de profesionalismo, en la actualidad no hay títulos escolarizados que tengan que ver con las tecnologías web. Por eso una forma de demostrar profesionalismo es mediante la validación de código, creando contenido web utilizando marcas semánticas y bien formadas, separación de estilo y contenido, etc. La validación se puede usar como una comprobación rápida para determinar si el código es el trabajo de un buen frontend o una sopa de etiquetas pirateada.

**Beneficios**

**¿Cómo los uso?**

Como todo software, ninguno es idéntico a otro, pero en la página del validador de CSS de la w3c hay un tutorial de cómo usarlo, y como es de la misma organización, tiene la misma interfaz y por lo tanto sabrás usar el validador de HTML:

<http://jigsaw.w3.org/css-validator/manual.html.es>

**¿Según cuáles validadores deben ser válidas mis páginas web?**

Tus archivos HTML y CSS deben ser validados por lo menos, con estos validadores:

Validador HTML y XML:<http://validator.w3.org/>

Validador CSS: http://jigsaw.w3.org/css-validator/

Validador de accesibilidad: <http://www.tawdis.net/>

**Bibliografía Web:**

**HTML5 y su historia:**

* <https://developer.mozilla.org/es/docs/HTML/HTML5>
* <https://developer.mozilla.org/es/docs/HTML/HTML5/HTML5_lista_elementos>
* <http://librosweb.es/libro/xhtml/capitulo_1/breve_historia_de_html.html>

**Buenas Prácticas**

**Prácticas generales:**

* <http://wiki.inf.utfsm.cl/index.php?title=Buenas_Practicas_de_Programaci%C3%B3n>

**De HTML5:**

* <https://www.codejobs.biz/es/blog/2016/01/19/buenas-practicas-en-html5>
* <http://www.danielgorostiaga.com/buenas-prcticas-para-la-maquetacin-en-html/>

**Cómo estructurar un proyecto de desarrollo.**

**Pautas generales:**

* <https://velneo.es/15-buenas-practicas-proyectos-desarrollo-software/>

**Pautas específicas: Ejemplos de modelos:**

* <https://sistemasvd.wordpress.com/2008/07/05/fases-del-proceso-de-desarrollo-del-software/>
* <https://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software>

**Código redundante**

* <https://es.wikipedia.org/wiki/C%C3%B3digo_redundante>
* <https://books.google.com.ar/books?id=gJrmszNHQV4C&pg=PA479&dq=redundant+code+meaning&hl=es-419&sa=X&ved=0ahUKEwjf_JyUoczaAhVEk5AKHQsvAjcQ6AEIJzAA#v=onepage&q=redundant%20code%20meaning&f=false>
* <https://es.quora.com/Qu%C3%A9-se-quiere-decir-con-c%C3%B3digo-elegante>

**Estándares de Programación**

* <https://es.scribd.com/document/36991779/Estandares-basicos-de-programacion>
* <http://yolopuedohacer.blogspot.com.ar/2010/06/estandares-de-programacion-manana-hoy-y.html>
* <http://www.net2client.net/manual/nuevomanual/Estandares_y_normas_de_programacion.htm>

**Ofuscación de Código:**

* <https://yuiblog.com/blog/2006/03/06/minification-v-obfuscation/>
* <https://blog.reaccionestudio.com/como-ofuscar-codigo-javascript-generador-online/>

**HTML sémantica:**

* <http://www.4rsoluciones.com/blog/html-la-importancia-del-codigo-semantico-2/>

**Meta-tags:**

* <https://www.1and1.es/digitalguide/paginas-web/desarrollo-web/los-meta-tags-mas-importantes-y-su-funcion/>
* <https://quiwiq.com/posicionamiento/importancia-meta-etiquetas-seo/952>
* <http://www.alhsis.com/las-meta-tags-y-su-importancia-en-el-seo/>
* <https://desarrolloweb.com/articulos/etiqueta-meta-viewport.html>
* <https://www.netvoluciona.es/blog/La-web-movil-y-la-utilidad-de-la-etiqueta--viewport--80>

**Validator:**

* <https://validator.w3.org/docs/why.html>
* <https://josewebmasterlibre.wordpress.com/2010/08/30/losvalidadores/>