AWAKELAB

Módulo 2: Fundamentos de Desarrollo Front-End

Desarrollador de aplicaciones Full Stack Python Trainee

MÓDULO 2 – FUNDAMENTOS DE DESARROLLO FRONT-END

2.5.- Bases del lenguaje JavaScript

2.5.1.- Bases del lenguaje JavaScript

2.5.1.1.- Breve historia de JavaScript

Hace más de dos décadas, el acceso a Internet se hacía generalmente con módems a una velocidad máxima de 28.8 kbps. Con eso se empezaron a desarrollar las primeras aplicaciones web y, por tanto, un sitio internamente era un panorama un poco complejo. Surgió entonces la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario, buscando entre otras cosas que, si el usuario no rellenaba correctamente un formulario web, se disminuía el tiempo de espera hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Un programador de Netscape llamado Brendan Eich, pensó que podría solucionar este problema adaptando otras tecnologías anteriores como ScriptEase al navegador Netscape Navigator en su versión 2.0, que se lanzaría en 1995; a este lenguaje le llamó LiveScript. Después Netscape firmó una alianza con la empresa Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript, solo por temas de marketing.

La primera versión de JavaScript fue un éxito, y en la siguiente versión del navegador Netscape, el “Navigator 3.0”, ya incorporaba la siguiente versión del lenguaje, la 1.1. En paralelo Microsoft lanzó JScript con su navegador Internet Explorer 3, pero era una copia de JavaScript con un nombre diferente, solo para evitar conflictos judiciales o legales.

2.5.1.2.- Relevancia de Javascript

JavaScript es un lenguaje de programación usado para procesar información y administrar archivos. Provee instrucciones que se ejecutan de forma secuencial para indicarle al sistema lo que se desea hacer, ya sea realizar una operación aritmética, asignar un valor a un dato, entre otras. Cuando el navegador encuentra este tipo de código en un documento, ejecuta las instrucciones y cualquier cambio realizado en el documento se muestra en pantalla.

Se puede hacer casi cualquier cosa con JavaScript. Se puede empezar con simples cosas como carruseles, galerías de imágenes, diseños fluctuantes, y respuestas a las pulsaciones de botones. Con más experiencia, se pueden crear juegos, animaciones 2D y gráficos 3D, aplicaciones integradas basadas en bases de datos, entre otras cosas. JavaScript por si solo es bastante compacto aunque bastante flexible, y los desarrolladores han escrito una gran cantidad de herramientas sobre del núcleo del lenguaje JavaScript, desbloqueando una gran cantidad de funcionalidad adicional con un mínimo esfuerzo. Esto incluye:

- Interfaces de Programación de Aplicaciones del Navegador (APIs): son aplicaciones construidas dentro de los navegadores que ofrecen funcionalidades tales como crear dinámicamente contenido HTML y establecer estilos CSS, hasta capturar y manipular un vídeo desde la cámara web del usuario, o generar gráficos 3D y muestras de sonido.
- APIs de Terceros: permiten a los desarrolladores incorporar funcionalidades en sus sitios de otros proveedores de contenidos como Twitter o Facebook.
- Marcos de trabajo y librerías de terceros: se pueden aplicar a cualquier HTML para que se puedan construir y publicar rápidamente sitios y aplicaciones.

JavaScript es una de las tecnologías web más atractivas del mercado, y a medida que un desarrollador conozca más funcionalidades, los sitios web creados entrarán en una nueva dimensión de energía y creatividad. Sin embargo, estar realmente cómodo con JavaScript es un poco más complejo que sentirse cómodo con HTML y CSS . El proceso debe ser paulatino, comenzando poco a poco y continuar trabajando en pasos pequeños y consistentes. A modo de ejemplo, se mostrará cómo añadir JavaScript básico en una página, insertando un texto "¡Hola Mundo!" en una etiqueta.

2.5.1.3.- Qué puede y no puede hacer en el contexto de un navegador

JavaScript es principalmente un lenguaje complementario, lo que significa que es poco común que una aplicación sea escrita por completo exclusivamente en JavaScript sin la ayuda de otros lenguajes como HTML y sin presentación en un navegador Web. Esto está cambiando con la llegada de productos como Node.js, que posibilitan crear aplicaciones en el lado del servidor basadas en JavaScript. Algunos productos de Adobe soportan JavaScript, y Windows 8 comienza a cambiar esto, pero el uso principal de JavaScript está en los navegadores.

JavaScript puede realizar muchas tareas en la parte cliente de la aplicación. Por ejemplo, se puede añadir la interactividad necesaria para una página Web mediante la creación de menús desplegables, transformar texto en una página, añadir elementos dinámicos a una página, o ayudar con la entrada de datos en un formulario.

Muchas de las operaciones que JavaScript no puede realizar son el resultado del uso de determinadas instrucciones limitadas a un entorno de navegador Web. Esta sección examina algunas de las tareas que JavaScript no puede llevar a cabo y algunas que JavaScript no debe realizar.

Entonces ¿qué no se puede hacer con JavaScript?

- **JavaScript no puede ser forzado en un cliente:** JavaScript descansa en otro interfaz o programa anfitrión para su funcionalidad. Este programa huésped suele ser el navegador Web del cliente, también conocido como agente de usuario. Debido a que JavaScript es un lenguaje del lado del cliente, se puede hacer sólo lo que el cliente permite que se haga.

- **JavaScript no puede garantizar seguridad de datos:** Debido a que JavaScript se ejecuta en su totalidad en el cliente, el desarrollador debe aprender a dejar el control. Después de que el programa está en el ordenador del cliente, éste puede hacer muchas cosas indeseables a los datos antes de enviarlos de nuevo al servidor. Al igual que con cualquier otro tipo de programación Web, nunca deberíamos confiar en los datos que vuelven desde el cliente. Incluso si hemos utilizado funciones JavaScript para validar el contenido de los formularios, todavía debemos validar estas entradas de nuevo cuando lleguen los datos al servidor. Un cliente con JavaScript desactivado podría devolver datos basura a través de un formulario Web. Si creemos, inocentemente, que nuestra función JavaScript del lado del cliente ya ha comprobado los datos para asegurarse de que son válidos, nos encontraremos con que se enviarán datos no válidos al servidor, provocando consecuencias imprevistas y posiblemente peligrosas.
- **JavaScript no puede atravesar dominios:** El desarrollador JavaScript debe también conocer la limitación que dicta que scripts que se ejecutan desde dentro de un dominio no tienen acceso a recursos de otro dominio de Internet, ni pueden afectar a scripts ni a datos de otro dominio. Por ejemplo, JavaScript puede ser utilizado para abrir una nueva ventana del navegador, pero el contenido de esa ventana es algo restringido a la secuencia de comandos que llama. Cuando una página de mi sitio Web (midominio.com) contiene JavaScript, esa página no puede acceder a ningún JavaScript ejecutado desde un dominio diferente, como google.com. Esta es la esencia de la política del mismo origen (en inglés Same-Origin Policy): JavaScript tiene que ser ejecutado u originado desde la misma ubicación.
- **JavaScript no hace servidores:** Desarrollando código del lado del servidor, por ejemplo, Visual Basic.NET o PHP (un acrónimo recursivo que significa PHP: Hypertext Preprocessor), podemos estar bastante seguros de que el servidor implementará ciertas funciones, como comunicarse con una base de datos o conceder acceso a módulos necesarios para la aplicación Web. JavaScript no tiene acceso a las variables del lado del servidor. Por ejemplo, JavaScript no puede acceder a bases de datos que se encuentran en el servidor. El código JavaScript se limita a lo que se puede hacer dentro de la plataforma sobre la que se está ejecutando el script, que suele ser el navegador.

2.5.1.4.- Cómo incorporar código Javascript en un documento HTML

Para agregar JavaScript a un proyecto debe hacer lo siguiente:

1.- Acceda al sitio de pruebas y cree una carpeta llamada 'scripts' (sin las comillas). Dentro de la nueva carpeta de scripts que acaba de crea, abra un nuevo archivo llamado main.js, y guárdelo en la carpeta scripts.
2.- A continuación, abra el archivo index.html e introduzca el siguiente elemento en una nueva línea, justo antes de la etiqueta de cierre </body>:

```
<script src="scripts/main.js"></script>
```

3.- Lo anterior hace básicamente un trabajo similar al del elemento <link> para CSS, aplicando un código JavaScript a la página a fin que ésta pueda actuar sobre el HTML, el CSS, o bien cualquier elemento en la página.
4.- Incluya el siguiente código en el archivo main.js:

```
var miTitulo = document.querySelector('h1');
miTitulo.innerHTML = '¡Hola Mundo!';
```

5.- Finalmente, asegúrese que ha guardado los archivos HTML y JavaScript, y abra el documento index.html en el navegador. El mensaje aparecerá en el elemento H1 declarado en el sitio.

En el caso anterior, el texto del título ha sido cambiado por "¡Hola Mundo!" usando JavaScript. Esto se logró usando la función querySelector() para obtener una referencia al título y almacenarla en una variable llamada “miTitulo”. Esto es muy similar a lo que se hizo anteriormente con CSS haciendo uso de selectores, en los cuales se desea hacer algo, y por tanto se tiene que seleccionar primero. Posterior a eso, se establece el valor de la propiedad innerHTML de la variable miTitulo, la que representa el contenido del título, y lo setea como "¡Hola Mundo!".

Finalmente, La razón por la que se ha puesto el elemento <script> casi al final del documento HTML es porque el navegador carga el HTML en el orden en que aparece en el archivo. Si se cargara primero JavaScript y se supone que debe afectar al HTML que tiene debajo, podría no funcionar, ya que ha sido cargado antes que el HTML sobre el que se supone debe trabajar. Por lo tanto, colocar el JavaScript cerca del final de la página es normalmente la mejor estrategia.

La razón por la que se ha puesto el elemento <script> casi al final del documento HTML es porque el navegador carga el HTML en el orden en que aparece en el archivo. Si se cargara primero JavaScript y se supone que debe afectar al HTML que tiene asociado, podría no funcionar ya que ha sido cargado antes que el HTML sobre el que se supone debe trabajar. Por lo tanto, colocar el JavaScript cerca del final de la página es generalmente la mejor estrategia.

2.5.1.5.- Selectores básicos: getElementById

El método getElementById () devuelve el elemento que tiene el atributo ID con el valor especificado.

Este método es uno de los métodos más comunes en HTML DOM y se usa casi cada vez que desea manipular u obtener información de un elemento en su documento. Devuelve nulo si no existe ningún elemento con el ID especificado.

Una identificación debe ser única dentro de una página. Sin embargo, si existe más de un elemento con el ID especificado, el método getElementById () devuelve el primer elemento en el código fuente.

2.5.1.6.- Obtención y manipulación de valores y textos de los elementos del DOM

El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. En resumen, es la representación de la página web en la memoria del navegador, a la que podemos acceder a través de JavaScript. El DOM es un árbol donde cada nodo es un objeto con todas sus propiedades y métodos que nos permiten modificarlo. Estas son algunas funciones que nos permiten acceder y modificar los elmentos del DOM:

Acceso a elementos del DOM

```
// Obtiene un elemento por id
document.getElementById('someid');

// Obtinee una lista con los elementos que tienen esa
clase
document.getElementsByClassName('someclass');

// Obtiene una HTMLCollection con los todos los elementos
'li'
document.getElementsByTagName('LI');

// Devuelve el primer elemento del documento que cumpla
la selección (la notación es como en CSS)
document.querySelector('.someclass');

// Devuelve una lista de elementos que cumplen con la
selección (notación como en CSS)
document.querySelectorAll('div.note, div.alert');
```

Acceder a hijos/padres de un elemento

```
// Obtener los hijos de un elemento
var elem = document.getElementById('someid');
var hijos = elem.childNodes;

// Su nodo padre
var padre = elem.parentNode;
```

Crear nuevos elementos en el DOM

```
// Para crear elementos llamamos a createElement con el
nombre del elemento
var nuevoH1 = document.createElement('h1');
var nuevoParrafo = document.createElement('p');

// Crear nodos de texto para un elemento
var textoH1 = document.createTextNode('Hola mundo!');
var textoParrafo = document.createTextNode('lorem
ipsum...');

// Añadir el texto a los elementos
nuevoH1.appendChild(textoH1);
nuevoParrafo.appendChild(textoParrafo);

// también podemos asignar directamente el valor a la
propiedad innerHTML
nuevoH1.innerHTML = textoH1
nuevoParrafo.innerHTML = textoParrafo

// los elementos estarían listos para añadirlos al DOM,
ahora mismo solo existen en memoria, pero no serán
visibles hasta que no los añadamos a un elemento del DOM
```

Añadir elementos al DOM

```
// seleccionamos un elemento
var cabecera = document.getElementById('cabecera');

// Añadir elementos hijos a un elemento
cabecera.appendChild(nuevoH1);
cabecera.appendChild(nuevoParrafo);

// También podemos añadir elementos ANTES del elemento
seleccionado

// Tomamos el padre
var padre = cabecera.parentNode;

// Insertamos el h1 antes de la cabecera
padre.insertBefore(nuevoH1, cabecera);
```

2.5.1.7.- Eventos básicos: onClick y onChange

Los eventos HTML DOM permiten que JavaScript registre diferentes controladores de eventos en elementos de un documento HTML.

Los eventos se usan normalmente en combinación con funciones, y la función no se ejecutará antes de que ocurra el evento (como cuando un usuario hace clic en un botón).

Evento onclick()

El evento onclick en JavaScript te permite como programador, ejecutar una función cuando se le da clic a un elemento.

```
<button onclick="miFunc()">Haz click</button>

<script>
  function miFunc() {
    alert('Se ha dado clic al botón!');
  }
</script>
```

En el ejemplo de arriba, cuando un usuario haga clic en el botón, verá una alerta en el navegador que muestra Se ha dado clic al botón!

El evento onclick se puede añadir también mediante código a cualquier elemento utilizando el siguiente ejemplo:

```
<p id="foo">Haz clic en este elemento.</p>

<script>
  let p = document.getElementById("foo"); // Encuentra el
elemento "p" en el sitio
  p.onclick = muestraAlerta; // Agrega función onclick al
elemento

  function muestraAlerta(evento) {
    alert("Evento onclick ejecutado!");
  }
</script>
```

Es importante notar que utilizando onclick solamente podemos agregar una función de escucha (listener function) que espera que algo suceda para capturarlo. Si quisieras agregar más, puedes utilizar addEventListener(), la cual es preferida para agregar capturadores de eventos.

Evento onchange()

El evento change se dispara para elementos <input>, <select>, y <textarea> cuando una alteración al valor de un elemento es confirmada por el usuario. A diferencia del evento input (en-US), el evento change no es disparado necesariamente por cada alteración al valor value del elemento.

Dependiendo del tipo de elemento siendo cambiado y la forma en que el usuario interactua con el elemento, el evento change dispara en un momento diferente:

- Cuando el elemento es :checked (ya sea dando click o usuando el teclado) para elementos <input type="radio"> y <input type="checkbox">;
- Cuando el usuario confirma el cambio explícitimante (por ejemplo, al seleccionar un valor de un menú desplegable <select> con un clic del ratón, al seleccionar una fecha de un selector de fecha de un elemento <input type="date">, al seleccionar un archivo en un selector de archivo por un elemento <input type="file">, etc.);
- Cuando un elemento pierde el foco después de que su valor haya sido cambiado, pero no confirmado (es decir, despues de editar el valor de un elemento <textarea> o <input type="text">).

```
<label>Elija un sabor de nieve:
    <select class="nieve" name="nieve">
        <option value="">Seleccione Uno …</option>
        <option value="chocolate">Chocolate</option>
        <option value="sardina">Sardina</option>
        <option value="vainilla">Vainilla</option>
    </select>
</label>

<div class="resultado"></div>
```

```
const selectElement = document.querySelector('.nieve');

selectElement.addEventListener('change', (event) => {
    const resultado =
document.querySelector('.resultado');
    resultado.textContent = `Te gusta el sabor
${event.target.value}`;
});
```

2.5.1.8.- Variables

En JavaScript, tanto var como let se usan para la declaración de variables, pero la diferencia entre ellos es que var tiene un alcance de función y let tiene un alcance de bloque. Se puede decir que una variable declarada con var se define en todo el programa en comparación con let.

A través del siguiente ejemplo se puede entender mejor el alcance de cada tipo de declaración.

```
<html>

<body>
```

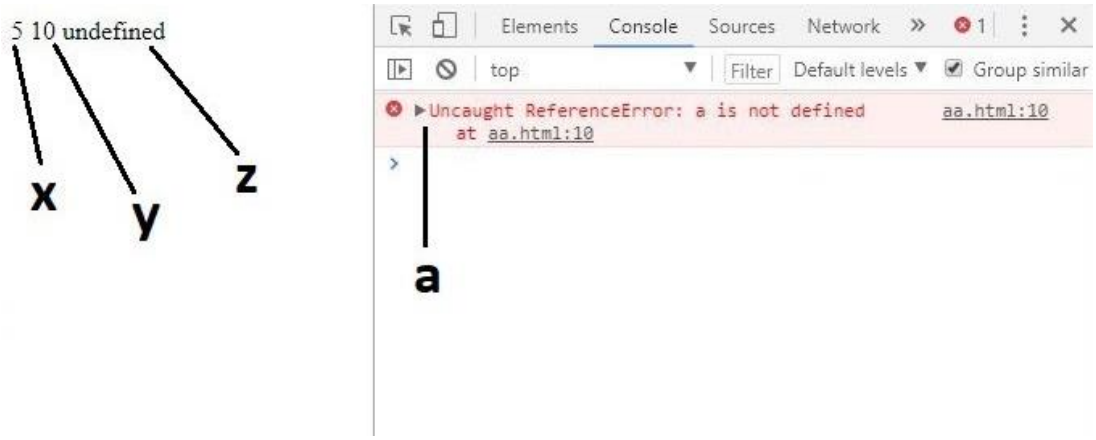
```
    <script>
        // Se llama a la variable x después de la
definición
        var x = 5;
        document.write(x, "\n");

        // Llamando a variable y después de la definición
        let y = 10;
        document.write(y, "\n");

        // Llamar a variable z antes de definición:
undefined (var)
        document.write(z, "\n");
        var z = 2;

        // Llamar a una variable a antes de definir:
error (let)
        document.write(a);
        let a = 3;
    </script>
</body>

</html>
```



En las ilustraciones anteriores se muestra la manera en la que se declaran variables en JavaScript. Cada tipo de declaración tiene una forma distinta de manejar las excepciones generadas por errores de sintaxis o alcance de variables.

Una constante corresponde a un valor que no cambia en el tiempo. Para definir una constante se usa el comando const, y si se intentan cambiar posterior a su declaración, se genera como respuesta un error.

```
const nacimiento = '18 .04.1982 ';
nacimiento  = '01 .01.2001 ';
// error, no se puede reasignar la constante!
```

2.5.1.9.- Expresiones aritméticas

Un operador es un símbolo que representa una acción sobre una o más variables. En términos generales, existen operadores unarios (aplican sobre una sola variable) y binarios (dos variables). Además, los operadores se pueden clasificar en virtud de su naturaleza o comportamiento en las siguientes categorías: aritméticos, de comparación y lógicos.

Dentro de los operadores aritméticos se pueden destacar los siguientes:

- Suma o adición (+)
- Resta o sustracción (-)
- División (/)
- Multiplicación (*)
- Resto o residuo (%)
- Exponenciación (**)
- Incremento (++)
- Decremento (--)

Los operadores de comparación, en tanto, comparan dos valores o variables, y retornan un valor lógico. Los siguientes son los operadores de comparación disponibles en JavaScript.

- == comprueba si el valor de la izquierda es igual al de la derecha.
- != comprueba si el valor de la izquierda es diferente al de la derecha.
- > comprueba si el valor de la izquierda es mayor que el de la derecha.
- < comprueba si el valor de la izquierda es menor que el de la derecha.
- >= comprueba si el valor de la izquierda es mayor o igual que el de la derecha.
- <= comprueba si el valor de la izquierda es menor o igual que el de la derecha.

Después de evaluar una condición, se devuelve como resultado un valor lógico, expresado como verdadero o falso. Esto permite trabajar con condiciones tales como si fueran valores y combinarlas para crear condiciones más complejas, tal como la que muestra el ejemplo anterior. JavaScript ofrece los siguientes operadores lógicos con este propósito.

- ! (negación) invierte el estado de la condición. Si la condición es verdadera, devuelve falso, y viceversa.
- && (y) comprueba dos condiciones y devuelve verdadero si ambas son verdaderas.
- || (o) comprueba dos condiciones y devuelve verdadero si una o ambas son verdaderas.

2.5.1.10.- Sentencias condicionales

En JavaScript existen dos tipos de expresiones condicionales: if y switch.

La sentencia “if” analiza una expresión y procesa un grupo de instrucciones en caso que la condición establecida por esa expresión sea verdadera. La instrucción requiere la palabra clave if seguida de la condición entre paréntesis y las instrucciones que se desea ejecutar si la condición es verdadera, esto entre llaves.

```
var capacitado = true;
var edad = 19;
if (edad < 21 && capacitado) {
   alert("Juan está autorizado");
}
```

Además, de forma similar a lo que se puede hacer en otros lenguajes, se puede asociar el comando “else”, permitiendo analizar escenarios alternativos.

```
var mivariable = 21;
if (mivariable < 10) {
  alert("El número es menor que 10");
} else {
  alert("El numero es igual o mayor que 10");
}
```

Si lo que se necesita es comprobar múltiples condiciones, en lugar de las instrucciones if else podemos usar la instrucción switch. Esta instrucción evalúa una expresión (generalmente una variable), compara el resultado con múltiples valores y ejecuta las instrucciones correspondientes al valor que coincide con la expresión. La sintaxis incluye la palabra clave switch seguida de la expresión entre paréntesis. Los posibles valores se indican usando la palabra clave case, de forma similar a como muestra el siguiente ejemplo.

```
var mivariable = 8;
switch(mivariable) {
  case 5:
    alert("El número es cinco");
    break;
  case 8:
    alert("El número es ocho");
    break;
  case 10:
    alert("El número es diez");
    break;
  default:
```

```
    alert("El número es " + mivariable);
}
```

2.5.1.11.- Funciones

Las funciones son bloques de código identificados con un nombre. La diferencia entre éstas y los bloques de código usados en los bucles y condicionales estudiados en la sección anterior, es que no hay que satisfacer ninguna condición; dicho de otra manera, las instrucciones situadas dentro de una función se ejecutan cada vez que se llama a ésta.

Las funciones se llaman, invocan o son ejecutadas escribiendo el nombre seguido de paréntesis. Esta llamada se puede realizar desde cualquier parte del código y cada vez que sea necesario, lo cual rompe completamente el procesamiento secuencial del programa. Una vez que una función es llamada, la ejecución del programa continúa con las instrucciones dentro de la función, sin importar dónde se localiza en el código, y solo devuelve a la sección del código que ha llamado la función cuando la ejecución de la misma ha finalizado.

Las funciones se declaran usando la palabra clave function, el nombre seguido de paréntesis, y el código entre llaves. Para llamar a la función se debe declarar su nombre con un par de paréntesis al final, tal y como se muestra en el ejemplo siguiente.

```
function mostrarMensaje() {
  alert("Soy una función");
}
mostrarMensaje();
```

Las funciones primero se declaran y posteriormente se ejecutan. El código del ejemplo anterior declara una función llamada mostrarMensaje() y luego la llama una vez. De forma similar a lo que pasa con las variables, el intérprete de JavaScript lee la función, almacena su contenido en memoria y asigna una referencia al nombre de la función. Cuando se invoca a la función por su nombre, el intérprete comprueba la referencia y la recupera desde la memoria, permitiendo llamar a la función todas las veces que sea necesario.

En JavaScript las instrucciones que se encuentran fuera de los límites de una función se consideran que están en un ámbito global, el cual es el espacio en el que se escriben las instrucciones hasta que se define una función u otra clase de estructura de datos. Las variables definidas en el ámbito global tienen un alcance global y, por lo tanto, se pueden usar desde cualquier parte del código, pero las declaradas dentro de las funciones tienen un alcance local, lo que significa que solo se pueden usar dentro de la función en la que se han declarado. Esta es otra ventaja de las funciones: son lugares especiales en el código donde se puede almacenar información a la que no se podrá acceder desde otras partes del código. Esta segregación ayuda a evitar generar duplicidades que pueden

conducir a errores, tales como sobrescribir el valor de una variable cuando el valor anterior aún era requerido por la aplicación.

```
var variableGlobal = 5;
function mifuncion(){
  var variableLocal = "El valor es ";
  alert(variableLocal + variableGlobal);
  // "El valor es 5"
}
mifuncion();
alert(variableLocal);
```

Otra manera de declarar una función es usando funciones anónimas. Las funciones anónimas son funciones sin un nombre o sin un identificador. Debido a lo anterior, se pueden pasar o entregar a otras funciones, o bien asignar a variables. Cuando una función anónima se asigna a una variable, el nombre de la variable es el que usamos para llamar a la función, tal como se muestra en el siguiente ejemplo.

```
var mifuncion = function(valor) {
  valor = valor * 2;
  return valor;
};
var total = 2;
for (var f = 0; f < 10; f++) {
  total = mifuncion(total);
}
alert("El total es " + total);  // "El total es 2048"
```

En el ejemplo anterior se declara una función anónima que recibe un valor, lo multiplica por 2 y devuelve el resultado. Debido a que la función se asigna a una variable, es posible usar el nombre de la variable para llamarla, por lo que después de que se define la función, se crea un bucle for que llama a la función mifuncion() varias veces con el valor actual de la variable total. La instrucción del bucle asigna el valor que devuelve la función de vuelta a la variable total, duplicando su valor en cada ciclo.

Finalmente, si bien existen las funciones estándar que pueden ser creadas por agentes externos, además se tiene acceso a funciones predefinidas por JavaScript. Estas funciones realizan procesos que simplifican tareas complejas; las siguientes son las que más se usan.

- isNaN(valor): devuelve true (verdadero) si el valor entre paréntesis no es un número.
- parseInt(valor): convierte una cadena de caracteres con un número en un número entero que se pueda procesar en operaciones aritméticas.

- parseFloat(valor): convierte una cadena de caracteres con un número en un número decimal que sea posible procesar en operaciones aritméticas.
- encodeURIComponent(valor): codifica una cadena de caracteres. Se utiliza para codificar los caracteres de un texto que puede crear problemas cuando se inserta en una URL.
- decodeURIComponent(valor): decodifica una cadena de caracteres.

2.5.1.12.- Cómo ejecutar código JavaScript en la consola

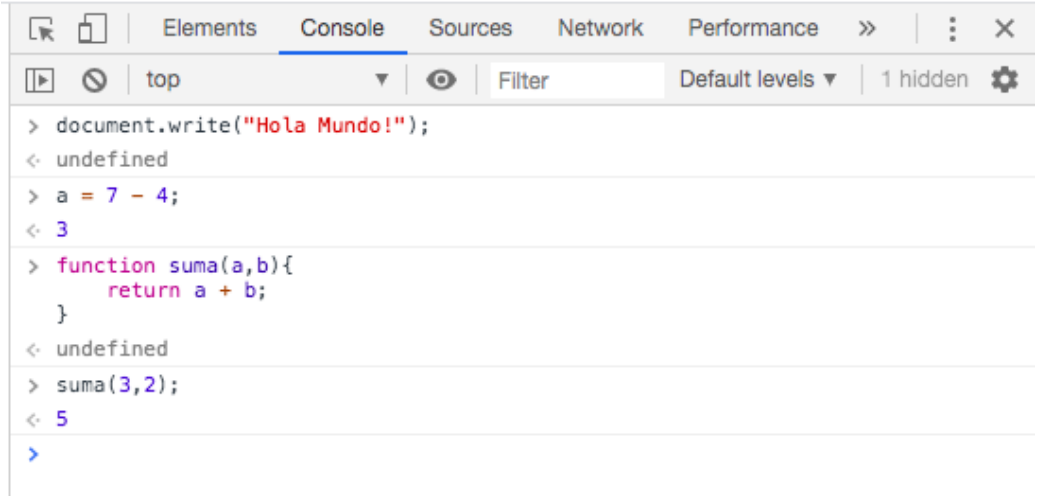
La consola de desarrollo es una herramienta que poseen los navegadores web comunes, que permite, entre otras cosas, poder ejecutar código asociado al lenguaje JavaScript. Otras de las características que posee, es poder recibir mensajes provenientes de la ejecución de determinados sitios, a fin de analizar su comportamiento.

En términos generales, la consola es una herramienta de tipo REPL, terminología que agrupa a todas las plataformas que permiten leer información, evaluar, imprimir y repetir acciones. Esta herramienta, entonces, lee el JavaScript que se escribe, evalúa el código y las expresiones incorporadas, imprime el resultado y vuelve al primer paso.

Para entender esta funcionalidad, se usará como referencia el navegador Google Chrome.

Debe seguir los siguientes pasos:

- Abra el navegador Google Chrome
- Presione los botones Command + Alt + J (Mac) o Control + Shift + J (Windows y Linux); se desplegará una ventana al costado derecho de la ventana, con la pestaña “Console” abierta.
- Escriba lo siguiente en la primera línea que esté habilitada en la consola: “document.write(“Hola Mundo”);” (sin comillas dobles). Verá que, como resultado, la página web desde la que desplegó la consola cambiará su texto.
- Además del despliegue de texto, puede realizar operaciones aritméticas sencillas. Ingrese el texto “a = 5 + 7” (sin comillas), y verá que el resultado se mostrará en la línea posterior.
- En general, la consola admite cualquier sentencia proveniente del lenguaje JavaScript. Como ejemplo, cree la función “suma”, que recibe dos números y despliega la suma de ambos.
- Finalmente, pruebe la función recién creada. Para ello escriba la sentencia “suma(3,2)” (sin comillas), lo cual invocará a la función recién creada, y mostrará el resultado obtenido.

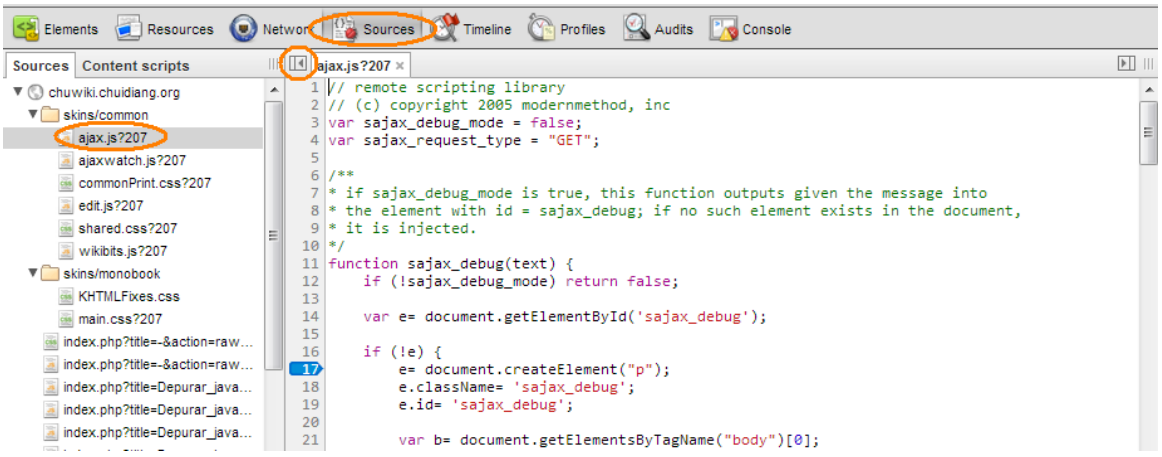


2.5.1.13.- Depurando el código JavaScript con la consola

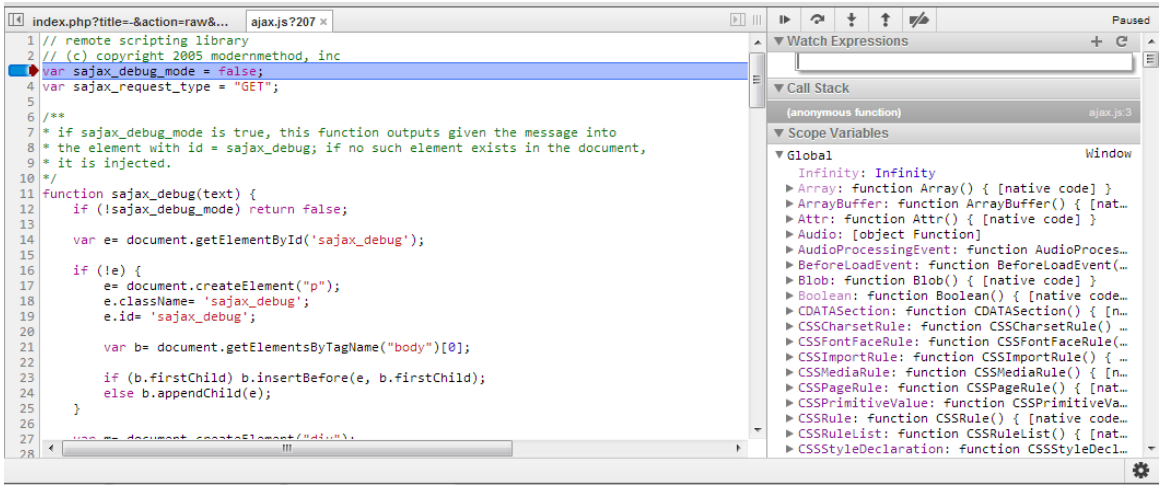
El navegador Google Chrome viene con herramientas de depuración de javascript integradas. Abrimos con él la página web que contiene el código javascript que nos interesa y pulsamos el botón de "personaliza y controla google chrome", unos puntitos que aparecen en la parte superior derecha del navegador. Se abre un menú y elegimos "Herramientas" -> "Consola de javascript". También se puede abrir rápidamente con la combinación de teclas Ctrl + Mayús + J.

Según lo tengamos configurado puede partirnos el navegador en dos, dejando en la parte inferior la consola de javascript, o bien puede abrirnos una ventana separada con la consola. Podemos separar o juntar dicha ventana pulsando el botón en la parte inferior izquierda cuyo tooltip pone "undock into separate window" o "dock into main window".

Para depurar el código javascript de la página, pulsamos el botón "Sources" y en la parte izquierda, debajo del botón "Elements" aparece una pequeña flechita que debmos pulsar. se abrirá un listado con todos los ficheros javascript que incluye la página. Seleccinando el de interés, vemos su código fuente (ver siguiente imagen).



En esa misma imagen hemos aprovechado para poner un breakpoint en la línea 17 sin más que hacer click sobre dicho número. Se ve como una flechita azul en ese número 17. Si ahora damos a recargar al navegador y el código javascript pasa por esa línea, se parará ahí. Por ejemplo, poniendo el breakpoint en la línea 3 y pulsando recargar la página en el navegador, vemos como el código javascript se para en la siguiente imagen.



Una vez parado, en la parte derecha tenemos botones para avanzar paso a paso, entrando o saliendo de llamadas a funciones. Podemos añadir un "watch" para ver cualquier variable que nos interese y podemos ver cualquier variable definida. Si colocamos el ratón sobre una de las variables del código que esté ya definida, podemos ver su valor. En la siguiente imagen hemos puesto el breakpoing en la línea 4 y una vez recargada la página y parado el debugger en esa línea, situando el ratón sobre la variable declarada en la línea 3 (la anterior), podemos ver su valor.

Referencias

[1] JavaScript: Manuales y cursos en línea
Referencia: https://www.javascript.com/

[2] J.D Gauchat “El Gran libro de HTML5, CCS3 y JavaScript”
3ª edición