

**TALENTO
DIGITAL**
INTELIGENCIA
HUMANA

Talento Digital para Chile:

Módulo 3 Fundamentos de Desarrollo Web

UN PROYECTO DE:

DESARROLLADO POR:



MÓDULO 3 – FUNDAMENTOS DE DESARROLLO WEB

3.3.- Contenido 3: Creación de algoritmos en JavaScript

Objetivo de la jornada

- Analizar distintas opciones de editores de texto para el edición de código
 - Entender el concepto detrás de JavaScript, su historia y las alternativas que existen en el mercado
 - Conocer la sintaxis básica de JavaScript, considerando la creación de variables
-

3.3.1.- Editores de texto

3.3.1.1.- Editores de texto recomendados

En clases anteriores, se mencionó que un editor de texto es un tipo de software diseñado para crear, modificar y almacenar archivos de texto. Un editor de texto podría ser una aplicación sin importancia para algunas personas, pero son herramientas que dan sentido a muchas organizaciones alrededor del mundo. Desde equipos de desarrollo a editoriales, los editores de texto y código están entrelazados con los procesos organizacionales. En el mundo de la programación es normal usar distintos editores dependiendo de la situación o tarea desarrollada. Sea que se esté escribiendo código Java, o simplemente tomando apuntes para un proyecto, existen muchas herramientas que permiten que esta tarea sea un poco más sencilla.

Dada la variedad de plataformas que existen en el mercado, es normal que toda persona tenga un editor de texto de preferencia, ya sea por facilidad de uso, tipo de sistema operativo o, simplemente, costumbre. Algunos de los editores de texto son excelentes para desarrolladores con mayor experiencia, mientras que otros son mejores para principiantes o escritores. También existen excelentes editores para colaboración, para compartir código en tiempo real y otras funcionalidades que facilitan las tareas recurrentes.

Dentro de los editores de texto más destacados, es necesario mencionar los siguientes:

- Visual Studio Code
- Atom
- Sublime Text
- Notepad++

A continuación se darán antecedentes de cada uno de ellos.

3.3.1.2.- Visual Studio Code

Anteriormente se mencionó que Visual Studio Code es un editor de código fuente creado por la empresa Microsoft, desarrollado para múltiples plataformas [1]. Este software combina la interfaz de usuario optimizada de un editor moderno con la asistencia y navegación de código enriquecido, y una experiencia de depuración integrada, todo lo anterior sin la necesidad de un IDE completo. Cuenta con herramientas de “**Debug**” hasta opciones para actualización en tiempo real del código en la vista del navegador y compilación en vivo de los lenguajes que lo requieran. Además de las extensiones, existe la posibilidad de optar por otros “temas” o plantillas de diseño, o bien que el usuario lo configure a su gusto.

Dentro de las ventajas que posee se pueden destacar:

- Es de uso liberado tanto comercial como privado.
- Posee soporte nativo para muchos lenguajes, incluyendo en ello los lenguajes asociados a Web: HTML, CSS y JavaScript.
- La interfaz se puede adaptar a gusto del usuario.
- Importante librería de temas o estilos visuales, permitiendo tener un trabajo más agradable al ojo humano.
- Cuenta con una nutrida comunidad de desarrolladores, quienes promueven mejoras continuas sobre la plataforma.

3.3.1.3.- Atom

Es un editor de texto de código abierto, denominado por sus creadores como “hecho para programadores” [2]. Tiene una comunidad importante de desarrolladores que permiten generar mejoras continuas, y varios plugins y temas que pueden ser incluidos. Un desarrollador experimentado no debiera tener problemas con este software ya que posee excelentes herramientas para el trabajo colaborativo, para edición de texto y de organización de contenidos.

Dentro de sus ventajas se destaca:

- Integración con GitHub, pudiendo crear desde ramas a fases en una sola interfaz
- Está presente para diversos sistemas operativos
- Posee autocompletado inteligente en el desarrollo de código fuente
- Instalación sencilla de nuevos paquetes y temas
- Múltiples paneles para edición de código

Atom es recomendable para aquellos desarrolladores que necesiten un editor de texto y una herramienta de colaboración sofisticada. Con este aplicativo es posible administrar proyectos con otros desarrolladores y ver cambios en el momento.

3.3.1.4.- Sublime Text

Sublime Text es un editor de texto usado por muchos desarrolladores en el mundo entero [3]. Tiene una versión gratuita con algunas limitaciones, y una versión de pago de alrededor de 80 dólares. Otra característica interesante de este programa es su tamaño; esto permite que use pocos recursos del equipo, convirtiéndolo en un excelente aliado para el desarrollo de código.

Otra característica interesante de este programa es el uso de atajos para la inserción de código, permitiendo al usuario presionar un par de teclas y pasar directamente a las opciones del menú. De esta manera, por ejemplo si un usuario desea ordenar elementos en un documento, no tendría que buscar en todo el código a fin de encontrar la funcionalidad.

Es considerada por muchos una de las mejores herramientas de edición de texto que existen en la actualidad. Esta afirmación cobra sentido al considerar los siguientes hechos:

- Posee una versión de prueba antes de tener que invertir en una licencia.
- Está disponible para los sistemas operativos más comunes
- Permite dividir ventanas, a actualizar más de un archivo al mismo tiempo
- Posee atajos que permiten escribir más código en menor tiempo
- Es altamente adaptable o configurable por cada usuario
- Posee una comunidad importante de desarrolladores, quienes constantemente dan a conocer nuevas funcionalidades del sistema a través de plugins

3.3.1.5.- Notepad++

Notepad++ es un editor de texto altamente conocido por muchas personas y muy utilizado [4]. Es de tipo gratuito, y está disponible para sistema operativo Windows. Al igual que otros editores de texto conocidos, busca ocupar poca memoria del computador del usuario, a fin de generar una experiencia óptima en el desarrollo de aplicativos. Está traducido a más de 80 idiomas lo que lo convierte en una plataforma muy utilizada mundialmente; además puede ser traducido a idioma nativo en caso que el idioma del usuario no esté disponible.

Otras características que lo hacen atractivo, es la posibilidad de resaltar código y la minimización de porciones de código, además de herramientas de búsqueda de datos y de reemplazo de cadenas de texto. Lo anterior sumado a otros aspectos la convierten en un editor de texto altamente recomendable:

- Su uso es completamente gratuito
- Se puede personalizar fácilmente, con herramientas enfocadas en desarrolladores avanzados
- Posee funciones de autocompletado, que permiten terminar funciones, parámetros y palabras reservadas sin tener que escribirlas una y otra vez
- Permite trabajar cada archivo en pestañas independientes
- Tiene una librería importante de plugins que mejoran la funcionalidad del editor, y que le permiten también conectarse con otros programas

Si una persona está introduciéndose al mundo del desarrollo web, esta herramienta es recomendable por su simpleza y adaptación a diversos escenarios. Además cuenta con todas las características necesarias para trabajar con lenguaje HTML, PHP o JavaScript.

3.3.1.6.- Consola de desarrollo

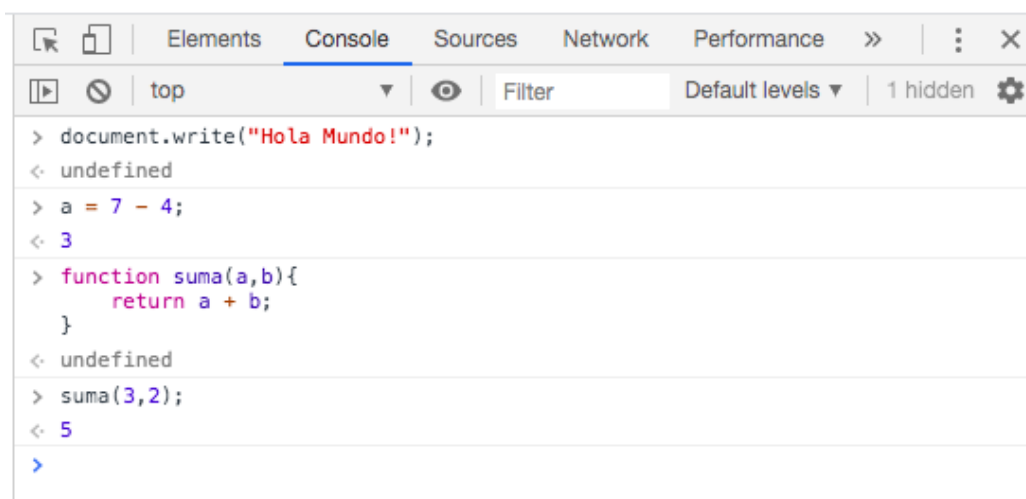
La consola de desarrollo es una herramienta que poseen los navegadores web comunes, que permite, entre otras cosas, poder ejecutar código asociado al lenguaje JavaScript. Otras de las características que posee, es poder recibir mensajes provenientes de la ejecución de determinados sitios, a fin de analizar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

- 1.- Abra el navegador Google Chrome
- 2.- 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.
- 3.- 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.

- 4.- 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.
- 5.- 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.
- 6.- 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.



```

> document.write("Hola Mundo!");
< undefined
> a = 7 - 4;
< 3
> function suma(a,b){
  return a + b;
}
< undefined
> suma(3,2);
< 5
>
  
```

Ilustración 1: Uso de consola en navegador Google Chrome

3.3.2.- JavaScript

3.3.2.1.- 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.

3.3.2.2.- Alcances 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.

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 crear, 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 hemos 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.

3.3.2.3.- Alternativas a JavaScript

De todas las opciones que existen en el mercado como “alternativas” a Javascript, tales como TypeScript o CoffeeScript, en su mayoría acaban compilando en dicho lenguaje. Además existen frameworks y librerías, pero igualmente están basadas en JavaScript o son derivados de lo mismo; ejemplo de ello son proyectos como React, AngularJS, Angular 2+, Vue.js y jQuery.

Sin embargo, si se desea un proyecto totalmente alternativo a este lenguaje sin uso del mismo, existe como opción WebAssembly . Se puede ejecutar desde varios navegadores, pero se sigue recomendando el uso de JavaScript y/o de sus derivados.

3.3.3.- Sintaxis básica de JavaScript

3.3.3.1.- Tipos de datos primitivos

JavaScript permite realizar numerosas tareas, desde calcular algoritmos complejos hasta procesar el contenido de un documento. Esto implica manipular valores, por tanto debe ser capaz de almacenar datos en memoria, concepto que hemos conocido como “variable”.

Las variables en JavaScript se declaran con la palabra clave **var** seguida del nombre que queremos asignarle. Si queremos almacenar un valor en el espacio de memoria asignado por el sistema a la variable, tenemos que incluir el carácter “igual” (=) seguido del valor, tal como se muestra acá.

```
var minumero = 2;
alert(minumero);
```

Ilustración 2: Declaración y uso de variables en JavaScript

Cuando se ejecuta el código anterior, el navegador reserva un espacio en memoria, almacenando el número 2 en su interior, y crea una referencia a ese espacio, asignando la referencia a la variable con nombre **minumero**. El método **alert()** permite desplegar un mensaje en el navegador.

Adicional a la declaración de variables indicada antes, existen otros tipos de datos primitivos en JavaScript. Cada uno es usado en contextos distintos y tiene formas de uso específicas. A continuación se muestra una tabla con el detalle y ejemplos de cada uno.

Tipo de dato	Definición	Ejemplo
Boolean	Representa una entidad lógica y puede tener dos valores: true, y false (verdadero y falso).	<pre>var valido = true; alert(valido);</pre>
Null	El tipo null tiene solo un valor posible: nulo.	<pre>var nulo = null; alert("La variable tiene el valor " + nulo);</pre>
Undefined	Una variable a la que no se le ha asignado un valor, llevará el valor undefined .	<pre>var algo = undefined; var otroalgo; alert("v1: " + algo + ", v2: " + otroalgo);</pre>

Tipo de dato	Definición	Ejemplo
Números	De acuerdo al estándar usado solo existe un valor numérico: valor de doble precisión de 64 bits. Para los enteros, por su parte, no existe un tipo específico.	<pre>> 42 / +0 < Infinity > 42 / -0 < -Infinity</pre>
Cadenas de texto	Para asignar texto a una variable, se debe declarar entre comillas simples o dobles.	<pre>var mitexto = "Hola Mundo!"; alert(mitexto);</pre>

Ilustración 3: Tipos de datos primitivos en JavaScript

3.3.3.2.- Variables y constantes

Anteriormente se analizó que una variable puede ser creada en JavaScript a través del comando **var**. Si bien esto es totalmente efectivo, se indica que esto es más bien parte de “la vieja escuela del lenguaje”, y hoy en día se promueve más el uso del comando **let**.

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>
```

Ilustración 4: Declaración de variables en JavaScript

5 10 undefined

x y z

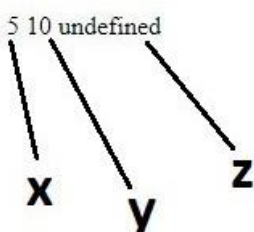
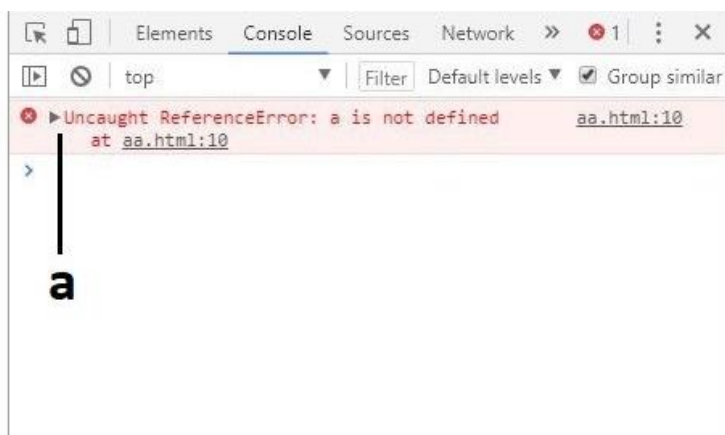



Ilustración 5: Resultado código ilustración 4

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!
```

Ilustración 6: Declaración de constantes en JavaScript

3.3.3.3.- Control de flujo y ciclos

Hasta ahora se han analizado ejemplos basados en instrucciones secuenciales, una después de la otra. En este tipo de programas el sistema ejecuta cada instrucción una sola vez y en el orden establecido, comenzando con la primera y siguiendo hasta llegar al final de la lista.

Los condicionales y bucles o ciclos tienen como misión romper esta secuencia. Los condicionales permiten ejecutar una o más instrucciones solo cuando se cumple una determinada condición, y los bucles dan la posibilidad de ejecutar un bloque de código o un grupo de instrucciones una y otra vez hasta que se cumpla una condición. JavaScript ofrece cuatro comandos para procesar código de acuerdo a condiciones que establece el programador: **if**, **switch**, **for** y **while**.

La manera más simple de comprobar una condición es con la instrucción **if**. Esta instrucción analiza una expresión y procesa un grupo de instrucciones si la condición establecida por esa expresión es verdadera. La instrucción requiere la palabra clave **if** seguida de la condición entre paréntesis y las instrucciones que queremos ejecutar si la condición es verdadera entre llaves.

Instrucción “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");
}
```

Ilustración 7: Uso de condicional en JavaScript

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");
}
```

Ilustración 8: Uso de condicional y alternativa en JavaScript

Instrucción "switch"

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);
}
```

Ilustración 9: Uso de instrucciones switch y case en JavaScript

Instrucción “for”

La instrucción **for** ejecuta el código entre llaves mientras la condición es verdadera. Usa la sintaxis **for (inicialización; condición; incremento)**. El primer parámetro establece los valores iniciales del bucle, el segundo parámetro es la condición que queremos comprobar y el último parámetro es una instrucción que determina cómo van a evolucionar los valores iniciales en cada ciclo.

```
var total = 0;
for (var f = 0; f < 5; f++) {
  total += 10;
}

alert("El total es: " + total); // "El total es: 50"
```

Ilustración 10: Uso de instrucción for en JavaScript

En el ejemplo anterior se realiza un ciclo de un total de cinco iteraciones, en donde cada una suma a una variable acumuladora un valor constante. Dado que la variable contador **f** se inicia en cero, y la condición no incluye el valor máximo, se logra dicha cantidad de iteraciones.

Instrucción “while”

La instrucción **while** corresponde a un ciclo que solo requiere la declaración de la condición entre paréntesis y el código a ser ejecutado entre llaves. El bucle se ejecuta constantemente hasta que la condición es falsa. A diferencia de su homólogo **for**, es útil cuando es posible determinar ciertas condiciones de términos, tales como el valor inicial del ciclo o el modo en que evolucionarán esos valores en cada iteración. Cuando esta información es poco clara, es posible hacer uso de la instrucción **while**.

```
var contador = 0;
while(contador < 100) {
  contador++;
}
alert("El valor es: " + contador); // "El valor es: 100"
```

Ilustración 11: Uso de instrucción while en JavaScript

En el ejemplo anterior el ciclo ya no depende de una cantidad de iteraciones, sino del cumplimiento de una condición de término. Se sabe, por supuesto, que el ciclo realizará cien iteraciones, que será reflejado en el resultado adjunto como comentario.

Instrucción “do while”

La instrucción **do while** ejecuta las instrucciones entre llaves y luego comprueba la condición, lo cual garantiza que las instrucciones se ejecutarán al menos una vez. La sintaxis es similar, solo se tiene que indicar antes de las llaves la palabra clave **do** y declarar la palabra clave **while** con la condición al final.

```
var contador = 150;
do {
  contador++;
} while(contador < 100);
alert("El valor es: " + contador); // "El valor es: 151"
```

Ilustración 12: Uso de instrucción do-while en JavaScript

En el ejemplo indicado anteriormente, el valor inicial de la variable `contador` es mayor de 99, pero debido a que se hace uso del bucle `do while`, la instrucción entre llaves se ejecuta una vez y, por lo tanto, el valor final de la variable `contador` es 151 ($150 + 1 = 151$).

3.3.3.4.- Operadores y comparadores

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.

3.3.3.5.- 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();
```

Ilustración 13: Declaración y llamado a una función

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);
```

Ilustración 14: Ámbito global versus local

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"
```

Ilustración 15: Ejemplo de función anónima

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.

3.3.3.6.- Implementando JavaScript

Siguiendo el mismo enfoque que CSS, el código JavaScript se puede incorporar al documento mediante tres técnicas diferentes: el código se puede insertar en un elemento por medio de atributos, incorporar al documento como contenido del elemento con la etiqueta **<script>**, o bien cargar desde un archivo externo. La

técnica segunda aprovecha atributos especiales que describen un evento, como un clic del ratón. Para lograr que un elemento responda a un evento usando esta técnica, todo lo que se tiene que hacer es agregar el atributo correspondiente con el código que se desea ejecutar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>JavaScript</title>
</head>
<body>
  <section>
    <p onclick="alert('Hizo clic!')">Clic aquí</p>
    <p>No puede hacer clic aquí</p>
  </section>
</body>
</html>
```

Ilustración 16: Definiendo JavaScript en línea

El atributo **onclick** es parte de una serie de atributos provistos por HTML para responder a eventos. La lista de atributos disponibles es larga y variada, pero éstos se pueden organizar en grupos dependiendo de sus propósitos. Los siguientes son los atributos más usados asociados con el ratón.

- **onclick:** responde al evento **click**. El evento se ejecuta cuando el usuario hace clic con el botón izquierdo del ratón. El lenguaje HTML ofrece otros dos atributos similares llamados **ondblclick** (el usuario hace doble clic con el botón izquierdo del ratón) y **oncontextmenu** (el usuario hace clic con el botón derecho del ratón).
- **onmousedown:** responde al evento **mousedown**, y se desencadena cuando el usuario presiona el botón izquierdo o el botón derecho del ratón.
- **onmouseup:** responde al evento **mouseup**, y se desencadena cuando el usuario libera el botón izquierdo del ratón.
- **onmouseenter:** responde al evento **mouseenter**, y se desencadena cuando el ratón se introduce en el área ocupada por el elemento.
- **onmouseleave:** responde al evento **mouseleave**, desencadenándose cuando el ratón abandona el área ocupada por el elemento.

- **onmouseover:** responde al evento **mouseover**, y este evento se desencadena cuando el ratón se mueve sobre el elemento o cualquiera de sus elementos hijos.
- **onmouseout:** responde al evento **mouseout**. El evento se desencadena cuando el ratón abandona el área ocupada por el elemento o cualquiera de sus elementos hijos.
- **onmousemove:** responde al evento **mousemove**, y se desencadena cada vez que el ratón se encuentra sobre el elemento y se mueve.
- **onwheel:** responde al evento **wheel**. Este evento se desencadena cada vez que se hace girar la rueda del ratón.

Los siguientes son los atributos disponibles para responder a eventos generados por el teclado; estos tipos de atributos se aplican a elementos que aceptan una entrada del usuario, como los elementos `<input>` y `<textarea>`.

- **onkeypress:** Este atributo responde al evento **keypress**. Este evento se desencadena cuando se activa el elemento y se pulsa una tecla.
- **onkeydown:** Este atributo responde al evento **keydown**. Se desencadena cuando se activa el elemento y se pulsa una tecla.
- **onkeyup:** Este atributo responde al evento **keyup**. Este evento se desencadena cuando se activa el elemento y se libera una tecla.

Finalmente, también es necesario indicar que se cuenta con otros dos atributos importantes asociados al documento:

- **onload:** Este atributo responde al evento **load**. El evento se desencadena cuando un recurso termina de cargarse.
- **onunload:** Este atributo responde al evento **unload**. Este evento se desencadena cuando un recurso termina de cargarse.

Anexo 1: Referencias

[1] Visual Studio Code

Referencia: <https://code.visualstudio.com/>

[2] Atom

Referencia: <https://atom.io/>

[3] Sublime Text

Referencia: <https://www.sublimetext.com/>

[4] Notepad++

Referencia: <https://atom.io/>

[5] JavaScript: Manuales y cursos en línea

Referencia: <https://www.javascript.com/>

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