Especialidad: Técnico en Programación

Módulo III. Submódulo2: Desarrolla aplicaciones que se ejecutan en el cliente.

Prof: Hilda Lucía Rodríguez Gómez

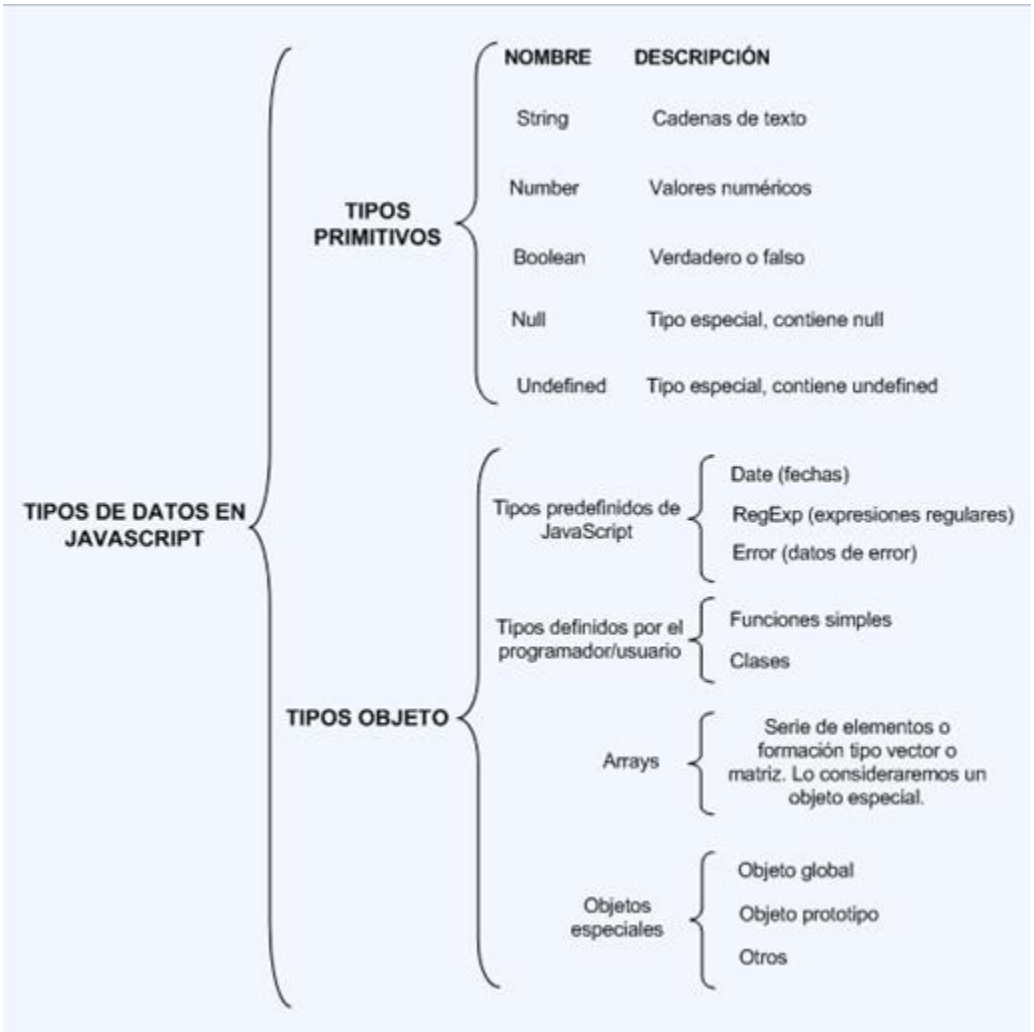
Competencia Profesional: Usa JavaScript para manejar eventos.

¿Qué es JavaScript? Lenguaje de programación ligero, multiplataforma, estructurado, orientado a objetos y eventos, el cual se puede aplicar a un documento HTML. Fue inventado por Brendan Eich.

Características:

- Seguro y fiable.
- Fácil uso y muy completo.
- Ligero y permite la elaboración de múltiples aplicaciones web.
- Se ejecuta en el navegador web (Cliente) sin necesidad de un servidor web y es compatible con la mayoría de navegadores.

TIPOS DE DATOS BÁSICOS EN JAVASCRIPT



OPERADORES

Aritméticos

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Substracción	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo: el resto después de la división	7 % 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	-a	Invierte el signo de un operando.

De Asignación:

Operador	Ejemplo	Expresión equivalente
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5

Relacionales:

Operador	Descripción
==	" Igual a" devuelve true si los operandos son iguales
===	Estrictamente "igual a" (JavaScript 1.3)
!=	" No igual a" devuelve true si los operandos no son iguales
!==	Estrictamente " No igual a" (JavaScript 1.3)
>	" Mayor que" devuelve true si el operador de la izquierda es mayor que el de la derecha.
>=	" Mayor o igual que " devuelve true si el operador de la izquierda es mayor o igual que el de la derecha.
<	" Menor que" devuelve true si el operador de la izquierda es menor que el de la derecha.
<=	"Menor o igual que" devuelve true si el operador de la izquierda es menor o igual que el de la derecha.

Lógicos:

Operador	Descripción
&&	Operador Y (AND). Verdadero si los dos operandos son verdaderos, falso en otro caso
||	Operador O (OR). Verdadero si al menos uno de los dos operando es verdaderos, falso en otro caso
!	Negación (NOT). Niega la variable. Si es verdadera, se evalúa a falso y viceversa

ESPECIFICACIONES GENERALES:

JS hace diferenciación entre mayúsculas y minúsculas. Para separar instrucciones podemos utilizar punto y coma (;) o un salto de línea; sin embargo, se recomienda terminar cada línea con punto y coma para mayor legibilidad del código; de igual manera, la "identación"-"sangria"-"margen" en el código, no es necesario aplicarla pero también se recomienda hacerlo. Las llaves {} se utilizan para englobar las sentencias que deban ejecutarse en caso de que se cumplan o no, según las expresiones lógicas dadas. Todas las estructuras de control se escriben en minúsculas (if, for, while, etc).

Comentarios: para comentar en una sola línea se usa la doble barra **// .** Los comentarios multilínea abren con **/*** y cierra con ***/**

Ejemplo:

```
<SCRIPT>
//Este es un comentario de una línea
/*Este comentario se puede extender
por varias líneas.
Las que quieras*/
</SCRIPT>
```

Variables

Los nombres de las variables aceptan caracteres alfanuméricos (números y letras), el carácter subrayado o guion bajo (_) y el carácter dólar. Dado que JS es sensitivo a mayúsculas y minúsculas, la variable “Edad” es diferente a la variable “edad”. Además de esto, los nombres de variables, no pueden comenzar por un carácter numérico, no podemos utilizar caracteres raros como el signo +, un espacio o un signo -, ni utilizar nombres de palabras reservadas de JS. Nombres admitidos para las variables podrían ser:

```
Edad;
_nombre;
$elemento;
paisDeNacimiento;
segundo_nombre;
```

Declaración: JS no obliga a declarar explícitamente las variables, pero se aconseja hacerlo utilizando la palabra reservada “var”, la cual, también permite asignar un valor inicial y declarar varias variables en la misma línea, siempre que se separen por comas. Ejemplos:

```
var operando1;
var operando2
var operando1 = 23;
var x = 2.8;
var nombre = “Pedro”;
var a,b,c;
var z=1, t=2;
var y = 5 + x;      //si x no ha sido definida marcaría error
```

Las variables de JS son NO TIPADAS, es decir pueden contener valores de cualquier tipo e ir cambiando según las necesidades del programa. Ejemplo:

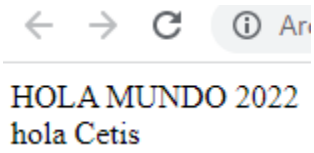
```
var x = 5           //crea la variable x y le asigna el valor inicial 5
x= “hola”           // asigna el string “hola”  a esa misma variable que tenía el valor de 5
```

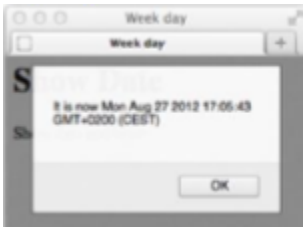
Si deseamos declarar una constante, se aplican las mismas reglas, pero se utiliza la palabra reservada “const”

SENTENCIAS PARA COMUNICAR MENSAJES

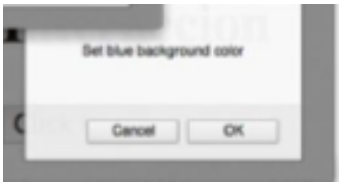
Podemos utilizar la instrucción **document.write,** la cual, nos permite escribir mensajes directamente en la página y las funciones alert(), confirm() y prompt(), las cuales proporcionan una interacción sencilla basa en “pop-ups”. Para desplegar un mensaje va entre comillas, si queremos agregar variables utilizamos el símbolo + , para agregar un salto de línea, la etiqueta
 va entre comillas. Ejemplos:

```
b=2022
document.write ("HOLA MUNDO " + b + "<br> hola Cetis");
```

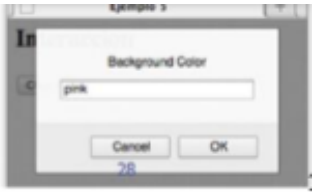




alert (“msg”); // presenta un pop-up con el “msg” establecido y un botón de OK



confirm(“msg”); // presenta un pop-up con el “msg” establecido y los botones OK Cancel



prompt(“msg”);

// presenta un pop-up con el “msg” establecido, pide un dato de entrada y los botones OK Cancel

La función prompt permite recibir información del usuario; Ejemplo:

```
var numero = prompt (“Introduzca un número”);
var x = 3 * numero;
```

ESTRUCTURAS DE CONTROL DE FLUJO

DECISIÓN if y switch

Sintaxis:

```
if (expresión) {
   //acciones a realizar en caso positivo
   //...
}
```

```
if (expresión) {
   //acciones a realizar en caso positivo
   //...
} else {
   //acciones a realizar en caso negativo
   //...
}
```

```
switch (expresión) {
    case valor1:
        Sentencias a ejecutar si la expresión tiene como valor a valor1
        break
    case valor2:
        Sentencias a ejecutar si la expresión tiene como valor a valor2
        break
    case valor3:
        Sentencias a ejecutar si la expresión tiene como valor a valor3
        break
    default:
        Sentencias a ejecutar si el valor no es ninguno de los anteriores
}
```

Iterativas (Bucles o Ciclos) for, while y do while

for

Sintaxis:

```
for (inicialización; condición; actualización) {
    //sentencias a ejecutar en cada iteración
}
```

Ejemplos:

```
var i
for (i=0;i<=10;i++) {
    document.write(i)
    document.write("<br>")
}
```

```
for (i=1;i<=6;i++) {
    document.write("<H" + i + ">Encabezado de nivel " + i + "</H" + i + ">")
}
```

while

Sintaxis:

```
while (condición){
    //sentencias a ejecutar
}
```

Ejemplo:

```
var color = ""
while (color != "rojo"){
    color = prompt("dame un color (escribe rojo para salir)","")
}
```

do while

Sintaxis:

```
do {
    //sentencias del bucle
} while (condición)
```

Ejemplo:

```
var color
do {
    color = prompt("dame un color (escribe rojo para salir)","")
} while (color != "rojo")
```

INCORPORANDO JAVASCRIPT EN HTML

EN LÍNEA

Esta es una técnica simple para insertar Javascript en nuestro documento que se aprovecha de atributos disponibles en elementos HTML. Estos atributos son manejadores de eventos que ejecutan código de acuerdo a la acción del usuario.

Los manejadores de eventos más usados son, en general, los relacionados con el ratón, como por ejemplo **onclick**, **onMouseOver**, u **onMouseOut**. Sin embargo, encontraremos sitios web que implementan eventos de teclado y de la ventana, ejecutando acciones luego de que una tecla es presionada o alguna condición en la ventana del navegador cambia (por ejemplo, **onload** u **onfocus**).

Usando el manejador de eventos **`onclick`**, un código es ejecutado cada vez que el usuario hace clic con el ratón sobre el texto que dice "Hacer Clic".

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
</head>
<body>
  <div id="principal">
    <p onclick="alert('hizo clic!')">Hacer Clic</p>
    <p>No puede hacer clic</p>
  </div>
</body>
</html>
```

El uso de Javascript dentro de etiquetas HTML está permitido en HTML5, pero, no es recomendable. El código HTML se extiende innecesariamente y se hace difícil de mantener y actualizar. Así mismo, el código distribuido sobre todo el documento complica la construcción de aplicaciones útiles.

EMBEBIDO

Para trabajar con códigos extensos y funciones personalizadas debemos agrupar los códigos en un mismo lugar entre etiquetas **<script>,** lo cual nos ayuda a organizar el código en un solo lugar, afectando a los elementos HTML por medio de referencias.

El elemento **`<script>`** y su contenido pueden ser posicionados en cualquier lugar del documento, dentro de otros elementos o entre ellos. Para mayor claridad, se recomienda colocar los códigos en la cabecera del documento y luego referenciar los elementos a ser afectados usando los métodos Javascript apropiados para ese propósito.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <script>
    function mostraralerta(){
      alert('hizo clic!');
    }
    function hacerclic(){
      document.getElementsByTagName('p')[0].onclick=mostraralerta;
    }
    window.onload=hacerclic;
  </script>
</head>
<body>
  <div id="principal">
    <p>Hacer Clic</p>
    <p>No puede hacer Clic</p>
  </div>
</body>
</html>
```

Existen tres métodos disponibles para referenciar elementos HTML desde Javascript:

- **getElementsByTagName** referencia un elemento por su nombre o palabra clave.
- **getElementById** referencia un elemento por el valor de su atributo **id**.
- **getElementsByClassName** es una nueva incorporación que nos permite referenciar un elemento por el valor de su atributo **class**.

Al colocar el código dentro de la cabecera del documento, debemos considerar que el código del documento es leído de forma secuencial por el navegador y no podemos referenciar un elemento que aún no ha sido creado.
En ejemplo, el código es posicionado en la cabecera del documento y es leído por el navegador previo a la creación del elemento **<p>** que estamos referenciando. Si hubiésemos intentado afectar el elemento **<p>** directamente con una referencia, hubiéramos recibido un mensaje de error anunciando que el elemento no existe. Para evitar este problema, el código fue convertido a una función llamada **mostraralerta()**, y la referencia al elemento **<p>** junto con el manejador del evento fueron colocados en una segunda función llamada **hacerclic()**.

Las funciones son llamadas desde la última línea del código usando otro manejador de eventos (en este caso asociado con la ventana) llamado **onload**. Este manejador ejecutará la función **hacerclic()** cuando el documento sea completamente cargado y todos los elementos creados. En el ejemplo, primero las funciones Javascript son cargadas (declaradas) pero no ejecutadas. Luego los elementos HTML, incluidos los elementos **<p>**, son creados. Y finalmente, cuando el documento completo es cargado en la ventana del navegador, el evento **load** es disparado y la función **hacerclic()** es llamada.

En esta función, el método **getElementsByTagName** referencia todos los elementos **<p>**. Este método retorna un arreglo (array) conteniendo una lista de los elementos de la clase especificada encontrados en el documento. Sin embargo, usando el índice **[0]** al final del método indicamos que solo queremos que el primer elemento de la lista sea retornado. Una vez que este elemento es identificado, el código registra el manejador de eventos **onclick** para el mismo. La función **mostraralerta()** será ejecutada cuando el evento **click** es disparado sobre este elemento mostrando el mensaje "hizo clic!".

ARCHIVOS EXTERNOS

Los códigos Javascript pueden crecer exponencialmente, para reducir los tiempos de descarga, incrementar nuestra productividad y poder distribuir y reusar nuestros códigos en cada documento sin comprometer eficiencia, se recomienda grabar todos los códigos Javascript en uno o más archivos externos y llamarlos usando el atributo **src**:

El elemento **<script>** carga los códigos Javascript desde un archivo externo llamado **micodigo.js**. Con esto, podremos insertar nuestros códigos en este archivo y luego incluir el mismo en cualquier documento de nuestro sitio web que lo necesite. Desde la perspectiva del usuario, esta práctica reduce tiempos de descarga y acceso a nuestro sitio web, mientras que para nosotros simplifica la organización y facilita el mantenimiento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <script src="micodigo.js"></script>
</head>
<body>
  <div id="principal">
    <p>Hacer Clic</p>
    <p>No puede hacer Clic</p>
  </div>
</body>
</html>
```