APLICACIONES WEB CON HTML5, CSS3 Y JAVASCRIPT

LUIS BASTO DÍAZ

Javascript

Introducción

Javascript es un lenguaje interpretado, no está relacionado con Java, aunque es un lenguaje que tiene un alto grado de similaridad, así como con: C#, C++, etc.

Basado en en ECMAScript (1) el cual se utiliza cada día más para para codificar aplicaciones y servicios del lado del servidor utilizando Node.js.

Se incorporaron Interfaces de programación de aplicaciones (APIs) por defecto en cada navegador para asistir al lenguaje en funciones elementales, tales como: Web Storage, Canvas, y otras.

La idea es hacer disponible poderosas funciones a través de técnicas de programación sencillas y estándares, expandiendo el alcance del lenguaje.

(1) https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Introduction#javascript_y_la_especificacion_ecmascript

Incorporar Javascript

Existen tres técnicas para incorporar código Javascript dentro de HTML.

Sin embargo, al igual que en CSS, solo la inclusión de archivos externos es la recomendada a usar en HTML5.

- En línea
- Embebido
- Archivos externos

Incorporar Javascript

En línea:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo 01 JS</title>
</head>
<body>
    <div id="principal">
    <p onclick="alert('clic al párrafo!')">Clic</p>
    <button onclick="alert('clic al botón!')">Clic</button>
    <input type="button" value="Hola" onclick="alert('clic al botón!')">
    <p>No se puede hacer clic</p> </div>
</body>
</html>
```

Incorporar Javascript

Embebido:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo 02 JS</title>
  <script>
     function alerta() {
       alert('hizo clic!’);
     }
     function hacerclic() {
        document.getElementsByTagName('p')[0].onclick = alerta;
     }
    window.onload = hacerclic;
  </script>
</head>
<body>
  <div id="principal">
    <p>Clic</p>
    <p>No se puede hacer clic</p>
  </div>
</body>
```

Incorporar Javascript

Archivo externo

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Ejemplo 02 JS</title>
    <script src="js/eje03_js.js"></script>
  </head>
<body>
  <div id="principal">
    <p>Clic</p>
    <p>No se puede hacer clic</p>
  </div>
</body>
</html>
```

Incorporar Javascript

Archivo externo

```
//eje03_js.js
function alerta() {
  alert('hizo clic!');
}
function hacerclic() {
  document.getElementsByTagName('p')[0].onclick = alerta;
}
window.onload = hacerclic;
```

Soporte a JavaScript

Algunos manejadores no soportan JavaScript o lo tienen deshabilitado.

El elemento <noscript> especifica el contenido alternativo a desplegar.

```
<script type="text/javascript">
var sumar = function(x,y) {
    return x + y;
};
alert(sumar( 5,8));
</script>
<noscript> Tu navegador no soporta JavaScript </noscript>
```

Tipos de valores

No define tipos de datos.

Define tres tipos de valores para los datos:

- Un valor primitivo
- Una función
- Un objeto

Valores primitivos

El valor primitivo es un dato que representa el nivel más bajo de la implementación del lenguaje, sus tipos pueden ser:

- number
- String
- boolean
- undefined
- null

Valores primitivos: number

Un valor numérico corresponde a un valor numérico: entero o flotante.

Es un número de doble precisión: 64 bits (formato IEEE 754)

Internamente son representados como valores de punto flotante.

Tener cuidado con el manejo de varios dígitos decimales porque podrían variar, lo mejor es redondear. Ejemplo: 0.1 + 0.2 no necesariamente será 0.3, puede ser 0.3000000004

Valores primitivos: number

Valores especiales soportados por los number:

- NaN: Not a number: realizar una operación con NaN resultará NaN
- Infinity: Representa el infinito positivo (>1.97E10308)
- -Infinity: Representa el infinito negativo(<-1.97E-10308)
- Undefined: No se ha asignado valor

Operadores aritméticos, lógicos, precedencia de operadores, :

- Ver: https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators

Valores primitivos: string

Colección de caracteres usados para representar texto.

Se encierra el texto entre comillas simples o dobles.

“Ejemplo encerrado en comillas dobles”

‘Ejemplo encerrado en comillas simples’

‘Ejemplo con “comillas dobles” encerradas en comillas simples’

“Ejemplo con ‘comillas simples’ encerradas en comillas dobles"

'este es un ejemplo \'con comilla simple\' ‘;

‘Este es un texto’ +
“que continúa en varias líneas." +

‘esta es la última línea...'

Valores primitivos: boolean, undefined y null

Boolean:

- Los tipos booleanos pueden contener los valores true y false.
- Se usan normalmente como resultado de operaciones relacionales: >, <, >=, <=, ==, ===

Undefined:

- Tiene un solo valor: undefined.
- De forma predeterminada, cuando una variable se declara, pero no se inicializa, se le asigna el valor undefined.

```
var contador;
console.log(contador);
console.log(typeof contador);
```

Valores primitivos: boolean, undefined y null

null:

Javascript define que null es un puntero a un objeto vacío.

```
var contador = null;
console.log(contador);
console.log(typeof contador);
```

Es una buena práctica asignar una variable a nulo (que mantenga un objeto en nulo) para que posteriormente se pueda verificar si el objeto es nulo o no utilizando la instrucción if de la siguiente manera:

```
if(obj != null) {
   // hacer algo
}
```

Valores primitivos: boolean, undefined y null

null:

- JavaScript define que nulo es igual a undefined con el operador ==.

```
console.log(null == undefined); //true
console.log(null === undefined); //false
```

Operadores == y ===

Operador ===

- El operador de estricta igualdad (===) revisa si dos operandos son iguales y produce un resultado Booleano.
- Si los operandos son de diferente tipo de valor, produce false
- Si ambos operandos son objetos, produce true solo si se refiere al mismo objeto.
- Si ambos operandos son de tipo null o ambos operandos son undefined, produce true.
- Si cualquier operando es de tipo NaN, produce false.

Operador ==

- Operador de igualdad regular (==) siempre considera que los operandos de distinto tipo de valor son diferentes.
- Si los operandos son de distinto tipo de valor, el operador == intenta convertir los valores a un mismo tipo de dato antes de compararlos.

Evaluación de Tipos

Las siguientes expresiones son verdaderas.

- null == undefined
- false == 0
- '' == 0
- '123' == 123

Las siguientes expresiones son falsas.

- null === undefined
- false === 0
- '' === 0
- '123' === 123

Variables

Los nombres de las variables (identificadores), se ajustan a ciertas reglas.

Un identificador de JavaScript debe comenzar con una letra, un guión bajo (_) o un signo de dólar ($). Los siguientes caracteres también pueden ser dígitos (0-9).

Dado que JavaScript distingue entre mayúsculas y minúsculas, las letras incluyen los caracteres "A" a "Z" (mayúsculas), así como "a" a "z" (minúsculas).

Algunos ejemplos de identificadores legales son

- Numero_clics, temp99, $credito, _name.

Declaración de variables

Con la palabra clave var.

- Ejemplo, var x = 42.
- Esta sintaxis se puede utilizar para declarar variables locales y globales, dependiendo del contexto de ejecución.

Con la palabra clave const o let.

- Ejemplo, let y = 13.
- Esta sintaxis se puede utilizar para declarar una variable local con ámbito de bloque.

También se puede simplemente asignar un valor a una variable.

- Ejemplo, x = 42.
- Esta forma crea una variable global no declarada.
- También genera una advertencia estricta de JavaScript.
- Las variables globales no declaradas a menudo pueden provocar un comportamiento inesperado.
- Por lo tanto, se desaconseja utilizar variables globales no declaradas.

Funciones

Una función es un conjunto de instrucciones que realiza una tarea o calcula un valor y devuelve una salida.

Para usar una función, se debe definir en algún lugar del ámbito desde el que se desea llamarla.

```
function sumar(x, y) {
   return x + y;
}
var a = sumar(5, 8);
```

Expresiones de Funciones

La declaración de función anterior sintácticamente es la declaración de la función.

Las funciones también se pueden crear mediante una expresión function.

Esta función puede ser anónima; no tiene por qué tener un nombre.

Por ejemplo, la función sumar se podría haber definido como:

```
const sumar = function (x, y) {
  return x + y;
};

var a = sumar(5, 8);
```

Expresiones de Funciones

Sin embargo, se puede proporcionar un nombre con una expresión function.

Proporcionar un nombre permite que la función se refiera a sí misma y facilita la identificación de la función en el seguimiento de la pila de un depurador:

```
const sumar = function sum(x, y) {
   return x + y;
};

console.log(sumar(3,5))
```

Expresiones de Funciones

Diferencias entre declarar una expresión función como constante (cons) y como variable (var)

```
var operacion = function (x, y) {
    return x + y;
};

var a = operacion(5, 8);
console.log(a);

operacion = function (x, y) {
    return x - y;
};

var b = operacion(5, 8);
console.log(b);
```

Expresiones de Funciones

Las expresiones function son convenientes cuando se pasa una función como argumento a otra función.

```
function myFun(fun, arr) {
  let result = []; // Crea un arreglo
  let i; // Declara una variable
  for (i = 0; i < arr.length; i++)
    result[i] = fun(arr[i]);
  return result;
}

const cubo = function (x) {
  return x * x * x;
};

let array = [2, 4, 6, 8, 10];
let valores = myFun(cubo, array)
console.log('valores:', valores);
```

Ámbito de una función

No se puede acceder a las variables definidas dentro de una función desde cualquier lugar fuera de la función (ámbito global).

La variable se define solo en el ámbito de la función.

Una función puede acceder a todas las variables y funciones definidas dentro del ámbito en el que está definida.

Javascript permite definir funciones anidadas, es decir, funciones dentro de funciones.

Ámbito de una función

Ejemplo:

```
// Las siguientes variables se definen en el ámbito global
var num1 = 25, num2 = 4, name = 'Luis';

// Esta función está definida en el ámbito global
function multiplicar() {
   return num1 * num2;
}

// Un ejemplo de función anidada
function adicionar() {
   var num1 = 2, num2 = 3;

function add() {
   return this.name + ' anotó ' + (num1 + num2);
}

return add();
}
```

Mensajes o alertas

El navegador provee las siguientes funciones para presentar datos y para obtener datos por parte del usuario:

Alert

- Despliega el mensaje en una ventana modal.
- El usuario debe hacer clic al botón para cerrar la ventana del mensaje.
- alert(“Hola Bienvenido”);

Prompt

- Pide un dato al usuario en una ventana modal con una caja de texto.
- Regresa el dato que el usuario escribió.
- var result = prompt(“El mensaje al usuario”, ‘valor default’);

Mensajes entrada/salida

Confirm

- Consulta la respuesta al usuario, Ok/Cancel en una ventana modal.
- Es true si el usuario presiona Ok, o false cuando presiona Cancel.
- Dando clic a ambos botones se cierra la ventana.
- var resultado = confirm(“¿Estás seguro de eliminar?”);

Las funciones prompt, confirm y alert pueden sobre escribirse, reemplazando su comportamiento por defecto, ya que el nombre de la función es una variable.

```
prompt = function(){
  return 'Hola';
}

console.log(prompt());
```

Conversiones

Función Number

- Intenta convertir el argumento objeto a un número.
- Si no se puede realizar la conversión, se regresa el valor NaN (Not a Number).

```
var edad = prompt('¿edad?', '');
alert('tu edad es: ' + Number(edad));

var edad = prompt('¿edad?', '');
alert('tu edad es: ' + Number(edad) + 1);

var edad = prompt('¿edad?', '');
alert('tu edad es: ' + (Number(edad) + 1) );
```

Conversiones

Función String

- Intenta convertir el argumento objeto a una cadena.

```
var x = 10;
var x = 20;
alert(String(x) + String(y));
```

Conversiones

Función isNaN

- Prueba si el argumento enviado es un número.

```
Var edad = prompt('¿edad?', '');
If(isNaN(edad))
        alert('El número no es válido');
else
        alert('Tu edad es'  + Number(edad));
```

Condicionales

```
window.onload = getEdad;

function getEdad() {
   var edad = prompt('Introduce tu edad', ’’);
   if (isNaN(edad)) {
     alert('Introduce un número válido’);
   } else {
      if (Number(edad) > 0 && Number(edad) < 18) {
         alert('Menor de edad!’);
      } else {
         if (Number(edad) >= 18 && Number(edad) < 60) {
            alert('Persona Adulta’);
         } else {
             alert('Adulto mayor’);
         }
        }
    }
}
```

Condicionales

```
function getColor() {
var color = prompt('¿Cuál es tu color favorito?', '');
switch (color) {
   case 'rojo’:
     alert('El color rojo es una buena opción!’);
     break;
   case 'negro’:
     alert('El color negro combina con todo!’);
     break;
   case 'blanco’:
     alert('Es genial este color’);
     break;
   default:
     alert('El color:' + color + ' no está en la lista’);
     break;
   }
}
```

Loops

```
var x = 10;
while (x > 0) {
    x--;
    alert("The value of x is " + x);
}
```

Loops

```
var retries = 0;
do {
   retries++;
   showLoginScreen();
} while (!authenticated() && retries < 3);

if (retries == 3) {
   alert('Too many tries');
}

function showLoginScreen(){
   alert('Login Screen’);
   authenticated();
}

function authenticated(){
  return false;
}
```

Loops

```
for (var counter = 0; counter < 10; counter++){
   alert('El contador es: ' + counter);
}
```

Loops

```
var numero = prompt('Número:', '');
var i = 2;
var esPrimo = true;
while (i < numero) {
    if (numero % i == 0) {
        esPrimo = false;
        break;
    }
    i++;
}
if (esPrimo) {
        alert(numero + ' es un número primo');
}
else {
        alert(numero + ' no es un número primo porque es divisible por ' + i);
}
```

Arrays

Colección de datos con una secuencia u orden.

Cada valor es un elemento del arreglo y es especificado por su índice.

Mantiene valores de diferentes tipos.

La longitud es dinámica, no es necesario especificar.

Arrays

Existen tres formas de crear y asignar datos a un arreglo

- Insertar elementos con un índice
- Formato de array condensado
- Array literal

Arrays

Insertar elementos con un índice

```
var ingredientes = new Array();
ingredientes[0] = 'azúcar';
ingredientes[1] = 'café';
ingredientes[2] = 'leche';


let scores = new Array();
let scores2 = Array(10);
let scores3 = [];
```

Arrays

Insertar elementos con un índice

```
var ingredientes = new Array();
ingredientes[0] = 'azúcar';
ingredientes[1] = 'café';
ingredientes[2] = 'leche';


let scores = new Array();
let scores2 = Array(10);
let scores3 = [];
```

Arrays

Formato de array condensado

```
var ingredientes2 = new Array('azúcar', 'café', 'leche');
```

Array literal

```
var ingredientes3 = ['azúcar', 'café', 'leche'];
```

Arrays

Para acceder a los valores del arreglo

```
for (let i=0; i < ingredientes.length; i++){
    console.log (ingredientes[i]);
}

for (let i of ingredientes){
    console.log(i);
}
```

Operaciones con arrays

concat: une dos o más arreglos y regresa un nuevo arreglo con todos los elementos.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var otros = ['pan', 'fruta'];
var todos = ingredientes4.concat(otros);
console.log(todos);
```

indexOf: localiza el elemento en el arreglo y regresa su índice.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var index = ingredientes4.indexOf('café');
```

Trabajando con Objetos

lastIndexOf: localiza el elemento desde el final del arreglo y regresa su índice.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var index = ingredientes4.lastIndexOf('azúcar');
```

join: crea una cadena de los elementos del arreglo delimitados por coma (se puede cambiar el separador).

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var cadena = ingredientes4.join();
```

Trabajando con Objetos

pop: Elimina y regresa el último elemento del arreglo.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var ultimo = ingredientes4.pop();
```

push: adiciona un nuevo elemento al final del arreglo y regresa la nueva longitud.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var longitud = ingredientes4.push('pan');
```

Trabajando con Objetos

reverse: invierte el orden de los elementos del arreglo y regresa una referencia al arreglo invertido.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
ingredientes4.reverse();
```

shift: elimina y regresa el primer elemento del arreglo.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var primero = ingredientes.shift();
```

Trabajando con Objetos

slice: regresa un nuevo arreglo que representa una parte del arreglo existente.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var pedazos = ingredientes.slice(1,3);
```

sort: ordena los elementos y regresa la referencia al arreglo.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
ingredientes4.sort();
```

Trabajando con Objetos

toString: Crea una cadena con los elementos del arreglo.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var cadena = ingredientes4.toString();
```

unshift: adiciona un nuevo elemento al principio del arreglo y devuelve la longitud.

```
let ingredientes4 = new Array('azúcar', 'café', 'leche');
var longitud = ingredientes4.unshift();
```