salesianos
DOMINGO SAVIO

Tema 3. Desarrollo de WEB. JAVASCRIPT

M04-Lenguajes de Marcas

¿QUÉ ES JAVASCRIPT?

- JavaScript es un lenguaje creado para dar vida a las páginas web.
  - Los programas que se ejecutan se llaman scripts. Se puede escribir directamente y se ejecutan cuando se carga la página.
  - No tiene nada que ver con el lenguaje Java

INICIAR UN SCRIPT

- Se inserta dentro de las etiquetas de HTML con la etiqueta <script></script> dentro de ellas se escribe el código de Javacript.
  - El elemento básico para que parezcan ventanas con mensajes es alert(‘Mensaje’);
  - Dentro de script todas las sentencias terminan en punto y coma
  - Se puede insertar en cualquier parte del código
  - Se puede hacer referencia a un archivo separado que termina en .js
    - <script src=”archivo.js"></script>
    - Si está establecido así el contenido interno de las etiquetas se ignora

SENTENCIAS

- Los comentarios se utilizan como C /* comentario */
- Para utilizar la versiones posteriores a 2009 se usa "use strict";
- JavaScript moderno usa “clases” y “módulos” que habilitan de forma automática las funcionalidades “modernas”
- VARIABLES
  - Se utiliza la palabra clave let
    - Let variable;
    - Podemos definir varias variables con un solo let var1,var2;
    - Let variable=valor;
  - En los scripts más viejo se puede utilizar var en vez de let
  - Los guiones no son permitidos en los nombres de las variables
  - Distingue entre mayúsculas y minúsculas
- CONSTANTES
  - const en vez de let, para declarar un valor que es inmutable
    - Const constate=19;

TIPO DE DATOS

- El tipo *number* representa tanto números enteros como de punto flotante.
  - Basta con asignarle un número a una variable
  - No representa valores mayor de (2^{53}-1)
- Infinity representa el Infinito matemático ∞
  - Es mayor que cualquier numero
  - alert(Infinity); o 1/0
- NaN: indica un error de cálculo.
- BigInt: representa los valores que no puede number
- String: representar cadenas
  - Comillas dobles: "Hola".
  - Comillas simples: 'Hola'.
  - Backticks (comillas invertidas): `Hola`. Los backticks son comillas de “funcionalidad extendida”. Nos permiten incrustar variables y expresiones en una cadena de caracteres encerrándolas en ${...}
    - let name = "John"; // incrustar una variable
    - alert(`Hola, ${name}!`); // Hola, John!
- Boolean (true o false)
- Valor null
- Valor undefined: valor sin definir. Se puede aplicar a cualquier variable
- Object: sirve para almacenar una colección de datos
- Typeof x: devuelve el tipo del argumento

ITERACCIÓN

- En JavaScript hay tres funciones que nos permiten interactuar con el usuario
  - Alert: muestra un mensaje y el usuario tiene que dar al botón Aceptar
  - Prompt: acepta dos argumentos. ;Muestra un titulo y un mensaje y permite aceptar o rechazar ese mensaje
    - prompt(title, [default]);
      - Title: texto para mostrar
      - Default: dato por defecto
  - Confirm: Muestra una ventana con una pregunta y dos opciones OK y CANCELAR
- Mensajes en body
  - document.writeln();

OPERACIONES MATEMÁTICAS

- Suma +
  - También concatena cadenas, con que una expresión sea un string convierte todo lo demás a string
- Resta -
- Multiplicación *
- División /
- Resto %
- Exponenciación **
- Modificaciones y asignaciones acortadas
  - n += 1; lo mismo que n = n + 1)
- Incremento y decremento
  - n++/++n y n--/--n
  - También se puede usar con otros operadores

COMPARACIONES

- Mayor/menor que: a > b, a < b.
- Mayor/menor o igual que: a >= b, a <= b.
- Igual: a == b
- Distinto a!=b
  - El resultado de las operaciones es booleano
- Si compara cadenas compara letra a letra
- Si compara cualquier tipo con números los convierte a números

CONDICIONALES

- if(...), ? evalúa la expresión dentro de los paréntesis. Las sentencias que se realiza cuando la operación es verdadera debe estar entre {}
- Tiene una parte else{}
- Concatenar condicionales Else if
- Operador ternario para las condicionales let result = condition ? value1 : value2;

OPERADORES LÓGICOS

- || (OR)
- && (AND)
- ! (NOT)
- Un doble NOT !!

BUCLES

- While: se ejecuta mientras se realiza una condición
  - while (condition)
  - {
  - // código
  - // llamado "cuerpo del bucle"
  - }
- Do…while: la condición se realiza al final del bucle
  - Do
  - { // cuerpo del bucle }
  - while (condition);
- For: se ejecuta un número de veces
  - for (begin; condition; step)
  - {
  - // (comienzo, condición, paso)
  - // ... cuerpo del bucle ...
  - }
- Se puede romper los bucles con un break

SWITCH

- switch(x)
- {
- case 'valor1': // if (x === 'valor1') ... [break]
- case 'valor2': // if (x === 'valor2') ... [break]
- default: ... [break]
- }

DECLARACIÓN DE FUNCIONES

- Las funciones se utilizan como C.
  - Function nombre(parametro1,parametro2…){
    - Contenido
  - }
- Para devolver un valor se utiliza return
  - return;→ es lo mismo que undefined
- Nomenclatura de funciones:
  - "get…" – devuelven un valor,
  - "calc…" – calculan algo,
  - "create…" – crean algo,
  - "check…" – revisan algo y devuelven un boolean, etc.

EXPRESIONES DE FUNCIÓN

- Por ejemplo tenemos esta función
  - Function nombre(){
    - Alert(“Mi nombre”);
    - }
  - Se va a reutilizar
- Podemos crear una funcion mediante una expresión de Función
  - Let nombre=function(){
    - Alert(“mi nombre”);
    - }
  - Debe ser definida antes de utilizarla

FUNCIONES FLECHA

- let func = (arg1, arg2, ..., argN) => expression;
  - Será
  - let func = function(arg1, arg2, ..., argN) { return expression; };a la expresión corta de poner:
- Ejemplo:
  - let sum = (a, b) => a + b;
  - let sum = function(a, b) { return a + b; };
- Funciones flecha multilinea
  - let sum = (a, b) => { // la llave abre una función multilínea
  - let result = a + b;
  - *return result; // si usamos llaves, entonces necesitamos un "return" explícito* };

FUNCIONES BÁSICAS

- **length**, calcula la longitud de una cadena de texto (el número de caracteres que la forman)
  - Por ejemplo, variable.length
- **+**: sirve para concatenar cadenas
- **toUpperCase(),** transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas
  - Variable.toUpperCase();
- **toLowerCase(),** transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas.
- **indexOf(caracter),** calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1.
- **lastIndexOf(caracter),** calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1.
- **substring(inicio, final),** extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final.
  - Si se indica un inicio negativo, se devuelve la misma cadena original
- **split(separador),** convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador

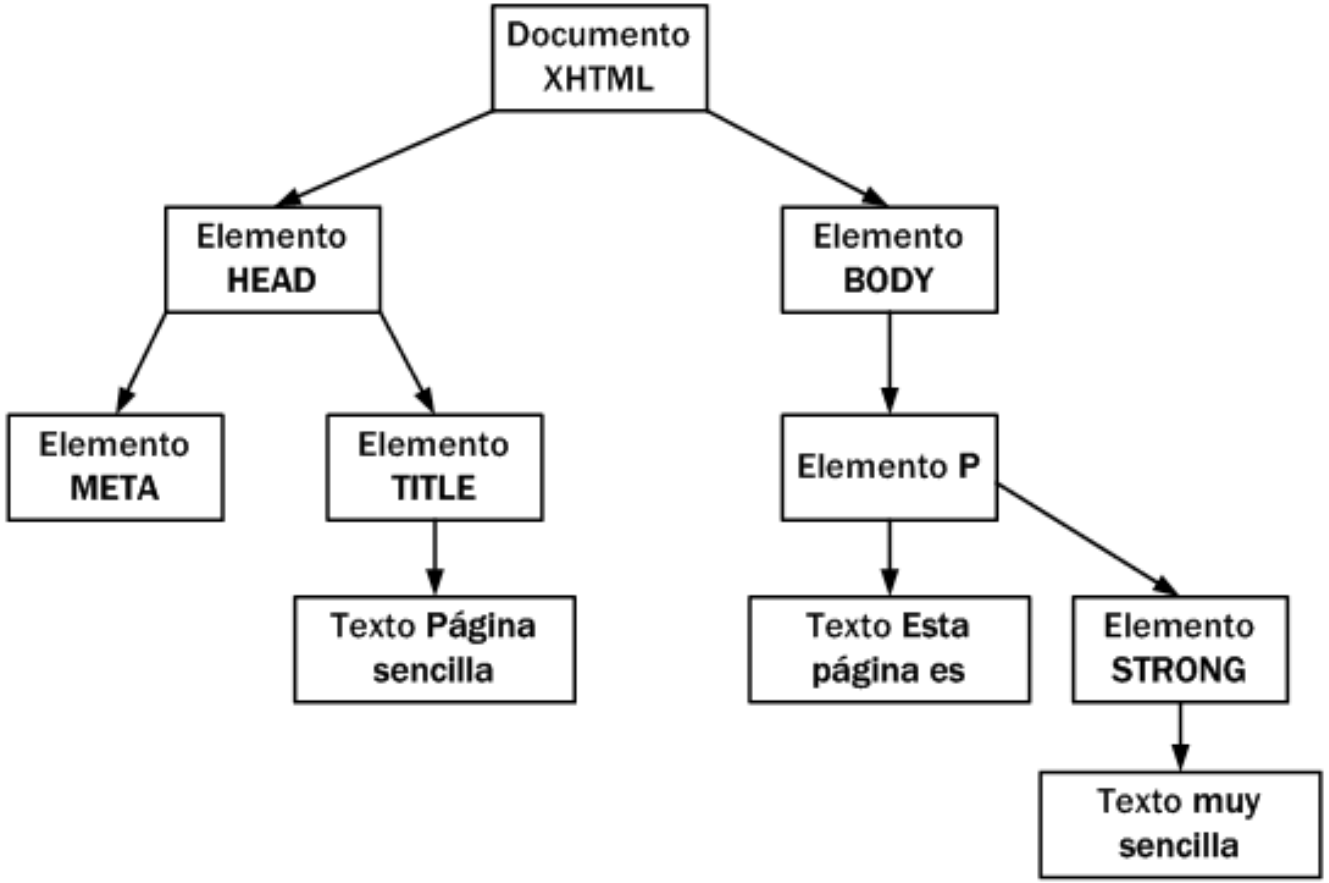
ARRAYS

- Los arrays son objetos similares a una lista cuyo objetivo es guardar datos dónde puedas acceder a las posiciones
  - let array = [“Dato1", “Dato2"]
  - **Acceder a un elemento de Array mediante su índice**
    - **Array[i]**
    - length, calcula el número de elementos de un array
    - concat(), se emplea para concatenar los elementos de varios arrays
    - join(separador), es la función contraria a split(). Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicador
    - pop(), elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento
    - push(), añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)
    - shift(), elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.
    - unshift(), añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)
    - reverse(), modifica un array colocando sus elementos en el orden inverso a su posición original

DOM

- La creación de Document Object Model o DOM es una de las innovaciones que a permitido mejorar las páginas web. Este nos permite utilizar las páginas XHTML como si fueran documentos XML.
- DOM transforma todos los documentos XHTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de la página web.
- La raíz del árbol de nodos de cualquier página XHTML siempre es la misma: un nodo de tipo especial denominado *"Documento"*.

ÁRBOL DE NODOS



TIPOS DE NODOS

- La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:
  - **Document**, nodo raíz del que derivan todos los demás nodos del árbol.
  - **Element**, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
  - **Attr**, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.
  - **Text**, nodo que contiene el texto encerrado por una etiqueta XHTML.
  - **Comment**, representa los comentarios incluidos en la página XHTML.
- Los otros tipos de nodos existentes que no se van a considerar son **DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.**

ACCESO A LOS NODOS

- **getElementsByTagName(nombreEtiqueta)** obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.
- **getElementsByName()** es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo name sea igual al parámetro proporcionado
- **getElementById() devuelve el elemento XHTML cuyo atributo id coincide con el p**arámetro indicado en la función. Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.