JAVASCRIPT

1. Introducción a JavaScript

- JavaScript es un lenguaje interpretado orientado a las páginas web.
- Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.
- Podrás mejorar tu página Web (efectos, animaciones, revisión de formularios, etc).

2. Fundamentos de JavaScript. Sintaxis.

- Incluir trozos de JavaScript dentro del código HTML de la página:

```
<p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
```

- Introducción de código JavaScript en las páginas en cualquier lugar (preferiblemente <head>):

```
<script type="text/javascript">
Programa JavaScript
</ script>
```

- Inclusión de ficheros externos con código JavaScript (suelen ir en el <head>):

```
<script type="text/javascript" src="fichero.js"></script>
```

Sintaxis

La sintaxis de un lenguaje define un conjunto de reglas que deben cumplirse al escribir código ejecutable.

Distinción entre mayúsculas y minúsculas

```
var num1;
var Num1;
```

Sintaxis con punto

El operador punto (.) proporciona una manera de acceder a las propiedades y métodos de un objeto:

```
window.alert("hola mundo");
```

Signos de punto y coma

Se puede utilizar el signo de punto y coma (;) para finalizar una sentencia.

```
var valor = 2;
```

Comentarios

- Para comentar una línea: //
- Para comentar varias: /* …… */

Las llaves: {…} permiten agrupar código.

Escritura de cadenas de texto en la página

document.write("Texto")

Cuadros de diálogo

alert("Texto")

prompt("Su color favorito es: ","Azul")

Palabras reservadas: break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with.

3. Variables y Tipos de datos

¿Qué es una variable?

Una variable es una unidad de almacenamiento de información. En ellas podremos ir guardando o cargando todos aquellos datos que necesitemos para realizar las acciones de nuestros programas.

Para declarar las variables lo haremos de la siguiente forma:

1. Usaremos números y letras del alfabeto inglés.
2. No puede empezar por un número.
3. Podemos usar "_" y $, pero no es recomendable.
4. El lenguaje distingue entre mayúsculas y minúsculas. Usar siempre minúsculas, para evitar malentendidos.

```
var mivariable;
```

Si hubiera que declarar más de una variable, se pueden declarar todas en una línea de código utilizando el operador coma (,) para separar las variables. Por ejemplo:

```
var operando1,operando2
```

Asignación de valores

Se usa el operador “**=**”:

```
var nombre="pepito";
```

Tipos de datos

Los datos se almacenan en variables, y pueden pasar de contener un tipo de dato a otro tipo (no es necesario indicar el tipo de dato que almacenará) y podemos operar con diferentes tipos de datos.

Se definen 3 tipos de datos en el lenguaje de programación:

- **Cadenas de caracteres**: son datos de texto y deben estar delimitados por comillas simples o dobles.
  ```
  var untexto = "Ser o no ser...";
  ```

- **Números**: Són datos numéricos que se asignan con el número en cuestión (siempre sin comillas).
  var unnumero = 17;
- **Booleanos**: un valor true (verdadero) o false (falso) (siempre sin comillas).
  var buenTiempo = true;

Arrays

Un array es una colección de variables. Los días de la semana se pueden agrupar en un array:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
```

Para acceder a cada uno de sus elementos hay que indicar su posición dentro del array (empiezan a contarse en el 0):

```
dias[0]; // "Lunes"

dias[5]; // "Sábado"
```

4. Operadores

¿Qué son los operadores?

Los operadores son símbolos especiales (o, en ocasiones, palabras) que se utilizan para realizar cálculos. Se utilizan principalmente en las operaciones matemáticas, pero también en la comparación entre valores. Por ejemplo:

El operador de suma (+) suma dos valores y obtiene como resultado una sola cifra:

```
var sum = 23 + 32;
```

Operadores Aritméticos

Son los utilizados para la realización de operaciones matemáticas simples como la suma, resta o multiplicación.

+ suma	- resta	* multiplicación	/ división
++ Incremento en una unidad		-- decremento en una unidad	

Operadores de asignación

Sirven para asignar valores a las variables. El más común es el "**=**".

Pero hay otros como:

x **+ =** y →(x = x + y)
x **- =** y → (x = x – y)
x *** =** y → (x = x * y)
x **/ =** y → (x = x / y)

mix=mix+4; *es lo mismo que:* mix +=4;

Ejemplo:
var ahorros = 7000 //asigna un 7000 a la variable ahorros
ahorros += 3500 //incrementa en 3500 la variable ahorros, ahora vale 10500
ahorros /= 2 //divide entre 2 mis ahorros, ahora quedan 5250

Operador de concatenación

El único operador de concatenación es el símbolo "**+**".

Ejemplo:

```
var cadena1 = "hola"
var cadena2 = "mundo"
var cadenaConcatenada = cadena1 + cadena2 //cadena concatenada vale "holamundo"
```

Operadores lógicos.

Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero (*true*) o un falso (*false*), y se utilizan para tomar decisiones en nuestros scripts.

! Operador **NO** o negación. Si era *true* pasa a *false* y viceversa.

&& Operador **Y**, si son los dos verdaderos vale verdadero.

|| Operador **O**, vale verdadero si por lo menos uno de ellos es verdadero.

Ejemplo: Si tengo hambre y tengo comida entonces me pongo a comer.

```
var tengoHambre, tengoComida, comoComida
tengoHambre = true;
tengoComida = true;
comoComida = tengoHambre && tengoComida
```

Operadores de comparación o condicionales

Se utilizan para tomar decisiones en función de la comparación de varios elementos y devuelven el valor booleano: true o false.

== Devuelve *true* si son iguales

!= Devuelve *true* sin son distintos

> Devuelve *true* si el 1er valor es mayor que el segundo

< Devuelve *true* si el 1er valor es menor que el segundo

>= Devuelve *true* si el 1er operando es mayor o igual que el segundo

<= Devuelve *true* si el 1er operando es menor e igual que el segundo

```
var i=5;

if (i==5) {
      window.alert("i es igual a 5");
}
```

5. Estructuras de control.

Los scripts vistos hasta ahora se iban ejecutando unos detrás de otros desde el principio hasta el final. Esto no tiene porqué ser siempre así.

En los programas generalmente necesitaremos hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir la misma línea de código una y otra vez.

Las Estructuras de control se dividen en 3 grupos:

a. **Estructuras condicionales**
b. **Bucles**
c. **Funciones**

Se consigue:

- Agrupar código: engloba bloques de código.
- Ordenar: controla el orden de ejecución.

a. Estructuras Condicionales:

Sirven para poder controlar el flujo del programa.

if..else

La sentencia condicional *if..else* permite comprobar una condición y ejecutar un bloque de código si dicha condición existe, o ejecutar un bloque de código alternativo si la condición no existe.

if es una estructura de control utilizada para tomar decisiones. Es un condicional que realiza unas u otras operaciones en función de una expresión.
La sintaxis de la estructura *if* es la siguiente.

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

También se pueden indicar acciones a realizar en caso de que la evaluación de la sentencia de resultados negativos.

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

switch

La sentencia **switch** evalúa una *expresión* y utiliza el resultado para determinar el bloque de código que debe ejecutarse. Los bloques de código empiezan por una sentencia **case** y terminan con una sentencia **break**.

```
switch (expersión) {
    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
}
```

b. Bucles for

Las sentencias de bucle permiten ejecutar un bloque específico de código repetidamente utilizando una serie de valores o variables.

El bucle FOR se utiliza para repetir unas instrucciones un determinado número de veces.

```
for (inicialización;condición;actualización)

{
   sentencias a ejecutar en cada interación
}
```

La inicialización: se ejecuta al comenzar la primera iteración del bucle (se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle).

La condición: contiene la condición que se debe cumplir para que continúe la ejecución del bucle.

La actualización: sirve para indicar los cambios que queramos ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.

c. Funciones

Podemos definir una función como una serie de instrucciones que englobamos dentro de un mismo proceso.

Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

(En algunos lenguajes de programación, las funciones también reciben los nombres de subrutinas o procedimientos.)

Las funciones pueden ser definidas por el usuario o bien vienen incorporadas en el propio lenguaje (métodos).

¿Cómo crear una función?

Una función empieza con la palabra clave **function**, seguida de:

■ El nombre de la función

■ Los parámetros (si los hay), en una lista delimitada por comas y escrita entre paréntesis

■ El cuerpo de la función (es decir, las instrucciones que deben ejecutarse cuando se invoca la función), escrito entre llaves

```
function nombrefuncion(parametros)
{
   instrucciones de la función    ...
}
```

¿Cómo llamar (invocar) a una función?

Para llamar a una función se utiliza su identificador (nombre) seguido del operador paréntesis ().

```
nombrefuncion();
```

Ejemplo:

```
<head>
<script>
function miFuncion(){
    document.write("Esto va bien")
}
</script>
</head>

<body>
<script>
miFuncion()
</script>
</body>
```

Pasar parámetros a las funciones

Los parámetros se usan para mandar valores a la función, con los que ella trabajará para realizar las acciones. Son los valores de entrada que recibe una función.

```
function bienvenida(mensaje){
    alert(mensaje);
}
bienvenida ("¿Qué tal el día?");
```

Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los parámetros separados por comas dentro de los paréntesis.

Funciones anónimas

La función también se puede definir sin un nombre asociado, y se conocen como funciones anónimas.

método tradicional:

```
function bienvenida(mensaje){
    alert(mensaje);
}

bienvenida("¿Qué tal el día?");
```

mediante una función anónima:

```
var miFuncion = function(mensaje){
    alert(mensaje);
}

miFuncion("¿Qué tal el día?");
```

Las funciones anónimas son ideales para los casos en los que se necesita definir funciones sencillas que solamente se utilizan una vez y para las que no es necesario crear una función tradicional con nombre.

6. Eventos

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas web (la pulsación de una tecla, pinchar o mover el ratón, seleccionar un elemento del formulario, redimensionar la ventana del navegador, etc.)

Tipos de eventos

Cada elemento o etiqueta HTML define su propia lista de posibles eventos que se le pueden asignar.

Los eventos más utilizados en las aplicaciones web son **onload** para esperar a que se cargue la página por completo, los eventos **onclick**, **onmouseover**, **onmouseout** para controlar el ratón y **onsubmit** para controlar el envío de los formularios.

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onload	La página se ha cargado completamente	<body>
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onsubmit	Enviar el formulario	<form>

[Tabla completa de eventos de JavaScript: http://www.librosweb.es/javascript/capitulo6/modelo_basico_de_eventos1.html]

Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado.

¿Dónde colocamos las funciones o código JavaScript a ejecutar?

1.- el código se incluye en un atributo del propio elemento HTML (*poco recomendable*):

```
<input type="button" value="pínchame y verás" onclick="alert('Gracias por pinchar');" />
```

2.- agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento HTML:

```
function muestraMensaje() {
alert('Gracias por pinchar');
}
<input type="button" value="pínchame y verás" onclick="muestraMensaje()" />
```

3.- usando manejadores semánticos, en los que no se mezcla el HTML con JavaScript:

```
// Función externa

function muestraMensaje() {
alert('Gracias por pinchar');
}

// Asignar la función externa al elemento

window.onload = function() {
    document.getElementById("pinchable").onclick = muestraMensaje;
}

// Elemento HTML

<input id="pinchable" type="button" value="pínchame y verás" />
```

7. DOM (Document Object Model)

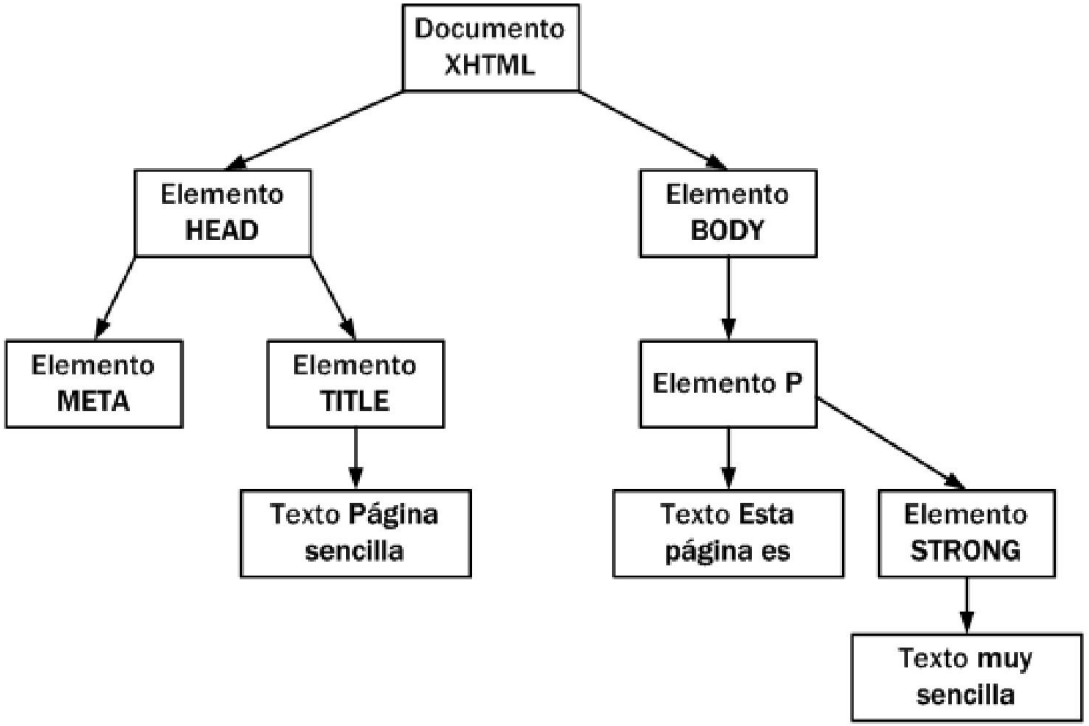
- El DOM (Modelo de Objetos de Documento) es una jerarquía de objetos predefinidos que describen los elementos de la página web que está mostrando el navegador.

- DOM permite a los programadores web acceder y manipular las páginas HTML.

- DOM transforma todos los documentos HTML en un conjunto de elementos llamados **nodos**, que están interconectados y que representan los contenidos de las páginas web y las relaciones que hay entre ellos.

Ejemplo de árbol de nodos generado automáticamente por DOM a partir del código HTML de la página:

```
<html>
<head>
<meta http-equiv="Content-
Type" content="text/html;
charset=utf-8" />
<title>Página sencilla</title>
</head>
<body>
<p>Esta página es <strong>muy
sencilla</strong></p>
</body>
</html>
```



En el esquema, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

La raíz del árbol de nodos de cualquier página es un nodo de tipo especial denominado "*Documento*" que representa a toda la página web.

A partir de ese nodo raíz, cada etiqueta HTML se transforma en un nodo de tipo "*Elemento*" (el primero es <html>). La conversión de etiquetas en nodos se realiza de forma jerárquica.

La transformación de las etiquetas HTML habituales genera dos nodos:
- nodo tipo "Elemento" (ej: <title>)
- nodo tipo "Texto" (ej: Página sencilla)

Tipos de nodos

▪ Document, nodo raíz del que derivan todos los demás nodos del árbol.

▪ Element, representa cada una de las etiquetas HTML.

▪ Attr, representa cada uno de los atributos de las etiquetas HTML.

▪ Text, nodo que contiene el texto encerrado por una etiqueta HTML.

El objeto en JavaScript: Propiedades y métodos

Los objetos se componen de un conjunto de valores (propiedades) y un conjunto de operaciones aplicados a esos valores (métodos). Son los métodos los que nos permiten modificar el estado de dicho objeto, es decir, el valor de sus propiedades.

mi_objeto.propiedad; por ejemplo: document.bgColor = "red";

mi_objeto. método(parámetros**);** por ejemplo: document.write("¡Hola mundo!");

Acceso directo a los nodos

La ventaja de DOM es que permite acceder y modificar todas las propiedades de los elementos HTML de forma directa.

Es importante recordar que el acceso a los nodos y su modificación solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página HTML se cargue por completo.

getElementsByTagName()

La función getElementsByTagName(nombreEtiqueta) obtiene todos los elementos de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

Ejemplo: cómo obtener todos los párrafos de una página HTML.

```
var parrafos = document.getElementsByTagName("p");
```

El valor que devuelve la función es un array con todos los nodos. Ahora, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

Y se podrían recorrer todos los párrafos de la página:

```
for(var i=0; i<parrafos.length; i++) {
var todoParrafo = parrafos[i];
}
```

getElementById()

La función getElementById(parametro) es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para **acceder directamente a un nodo** y poder leer o modificar sus propiedades.

Devuelve el elemento HTML cuyo atributo **id** coincide con el parámetro indicado en la función.

```
var midiv = document.getElementById("cabecera");
……
<div id="cabecera"><a href="#" id="logo">...</a></div>
```

Modificar HTML mediante JavaScript

Mediante la propiedad **innerHTML** accedemos al contenido de texto de un elemento y lo modificamos:

```
<div id="midiv"></div>

document.getElementById("midiv").innerHTML = "Contenido para la capa";
```

Modificar CSS mediante JavaScript

Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo **style** seguido de la propiedad CSS con su nuevo valor:

```
document.getElementById("miparrafo").style.fontSize="24px";
```

[Tabla de conversión de las propiedades CSS a sus referencias de JavaScript:

http://refugioantiaereo.com/2006/03/conversion-de-propiedades-css-a-referencias-javascript]

O también a través de la propiedad **className** podemos asignar una clase a un elemento HTML:

```
document.getElementById("midiv").className = "miclase";
```

```
.miclase { position: absolute;
 left: 265px;
 top: 47px;
 width: 842px;
 height: 117px;
 z-index: 1;
 background-color: #990000;
}
```

8. Tareas de Refactoring

La refactorización (del inglés *refactoring*) es una técnica para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

Las tareas genéricas en el refactoring:

- Limpiar código muerto (código que nunca se ejecuta).
- Eliminar funcionalidades duplicadas.
- Encapsular funcionalidades beneficiando la reutilización de código.
- Actualizar librerías y código a nuevas versiones disponibles de las tecnologías utilizadas.
- Seguir los principios básicos y metódicos.
- Optimizar el rendimiento.

Ventajas de refactorizar:

- Código óptimo, mantenible y más claro. Agilizando el desarrollo sobre él.
- Código encapsulado y disponible para la reutilización. Evitando desarrollar la misma funcionalidad más de una vez.
- Un correcto rendimiento de la aplicación.
- Menos bugs/incidencias y vulnerabilidades (seguridad).
- Reducción de costes.