PROGRAMACIÓN MULTIPLATAFORMA

JavaScript

Lógica de programación

- La lógica forma parte nuestra.
- Somos capaces de diseñar de forma lógica.
- No necesitamos conocimientos de programación para que nos funcione la lógica.
- La lógica en buena parte será la responsable del funcionamiento bueno o malo del programa.
- Programar es traducir la lógica a un lenguaje que el sistema entienda.

Desarrollo web

- En desarrollo web, nuestro código, también llamado *script* se ejecutará en dos partes.
  - En el lado del cliente
  - En el lado del servidor

Lado del cliente

- La programación en el cliente acompañará a un documento HTML o estará incluido en él.
- El programa se ejecuta en la máquina del cliente cuando se carga el documento, o en algún otro instante, como por ejemplo cuando se activa un vínculo.

Lado del cliente. Capacidades

- Los scripts ofrecen la posibilidad de extender los documentos HTML de maneras activas e interactivas. Por ejemplo:
  - Pueden evaluarse los scripts cuando el usuario interactúe con la web para modificar los contenidos del documento dinámicamente.
  - Los scripts pueden acompañar a un formulario para procesar los datos a medida que éstos se introducen.

Lado del cliente. Capacidades

- Nos pueden servir para validar información.
- Los scripts pueden ser llamados por eventos que afecten al documento, como la carga, la descarga, el movimiento del foco sobre los elementos, los movimientos del ratón, etc.
- Los scripts pueden ser vinculados a controles de formulario (por ejemplo botones) para desencadenar cambios en las páginas.

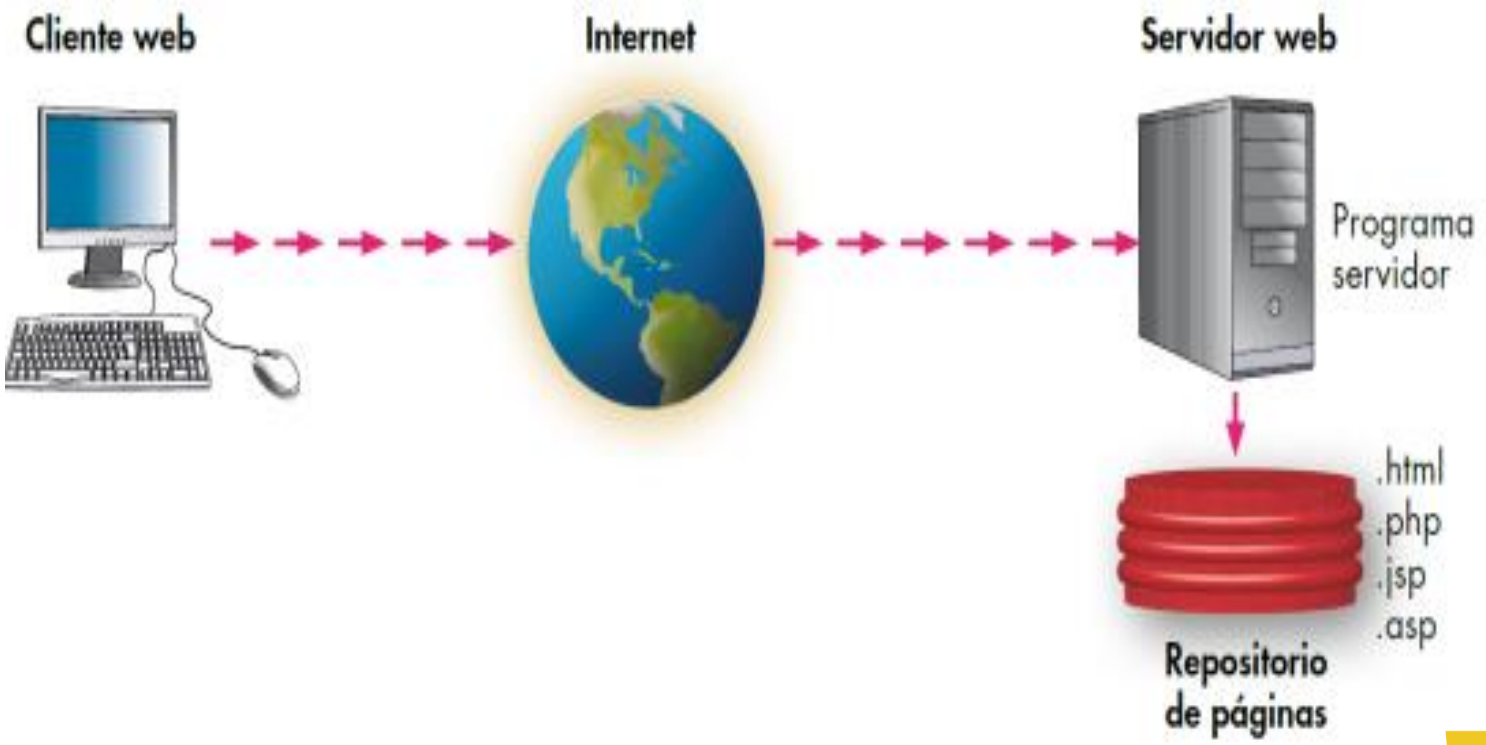
Lado del servidor

- El desarrollo en el lado del servidor no se ejecuta en el navegador del usuario que está viendo una web, sino que se ejecuta en el servidor donde está alojada.
- Los scripts del lado del servidor ofrecen posibilidades más complejas, como por ejemplo:

Lado del servidor. Capacidades

- Acceso a datos alojados en una base de datos.
- Sistemas de autenticación.
- Consultas de datos, artículos, carrito de la compra…

Resumen


Cliente web
Internet
Servidor web
Programa servidor
.html
.php
.jsp
.asp
Repositorio de páginas

Características de JavaScript: Orientado a eventos.

- Un evento es un suceso que ocurre en el sistema, generalmente en nuestro programa.
- Por ejemplo, un usuario desencadena un evento cuando pulsa en un botón esperando que ocurra algo, como enviar sus datos previamente rellenados en un formulario.
- El manejador de eventos de la plataforma que soporta el lenguaje es quien cumple dicha función.

Características de JavaScript: Interpretado

- Un lenguaje interpretado es un lenguaje de programación que ejecuta las instrucciones directamente, sin una previa compilación del programa a instrucciones en lenguaje máquina.
- El intérprete ejecuta el programa directamente, traduciendo cada sentencia y ejecutándola.

Características de JavaScript: Sintaxis (I)

- **No se tienen en cuenta los espacios en blanco y las nuevas líneas:** Como en HTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor.
- **Se distinguen las mayúsculas y minúsculas:** Si en JavaScript se intercambian mayúsculas y minúsculas en variables o palabras reservadas el script no funciona.

Características de JavaScript: Sintaxis (II)

- **No se define el tipo de las variables:** Al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
- **No es necesario terminar cada sentencia con el carácter de punto y coma (;):** No es necesario, aunque es una buena práctica.

JavaScript: Sintaxis (III)

- **Se pueden incluir comentarios**: Los comentarios se utilizan para añadir información en el código fuente del programa.
- Aunque no se ven en pantalla, los comentarios se envían al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.
- Los comentarios se pueden registrar en una sola línea con dos barras (//) o en varias líneas con /* y */.

¿Scripts en body o en head?

- La diferencia radica en los tiempos de ejecución.
- La recomendación más extendida consiste en enlazarlo o incluirlo al final de la página, justo antes de la etiqueta </body> de cierre.
- Las buenas prácticas de empresas líderes en el desarrollo web así lo aconsejan. Yahoo!, sobre el rendimiento de los scripts en las aplicaciones web indica lo siguiente al respecto:
- *"El problema de los scripts es que mientras se descargan, bloquean la descarga de otros contenidos importantes de la página, como por ejemplo las imágenes."*

Incluir JavaScript en HTML en un archivo externo (I).

- El código JavaScript se puede incluir en un archivo externo de tipo JavaScript.
- Un documento HTML se vincula al archivo mediante la etiqueta **<script>**. Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento HTML puede enlazar tantos archivos JavaScript como necesite.
- Además del atributo **type** que especifica el lenguaje del script, este método requiere definir el atributo **src**, que es el que indica la URL correspondiente al archivo JavaScript que se quiere enlazar.

Incluir JavaScript en HTML en un archivo externo (II).

- Cada etiqueta <script> solamente puede enlazar un único archivo, pero en una misma página se pueden incluir tantas etiquetas <script> como sean necesarias.
- Los archivos de tipo JavaScript son documentos normales de texto plano con la extensión **.js**.

Incluir JavaScript en HTML en un archivo externo (III).

- Esta manera es la mejor para incluir código JavaScript en un documento porque separa totalmente el código de marcado del código JavaScript, lo que hace todo más mantenible.
- Así podremos vincular el mismo código a más páginas y las modificaciones las haremos sólo en un sitio.

La etiqueta HTML <noscript>

- Algunos navegadores permiten bloquear JavaScript.
- Hay quien tiene la percepción de que así la navegación es más segura.
- En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página.
- La etiqueta **<noscript>** permite gestionar este aspecto.

Variables

- Una variable es un elemento que se emplea para almacenar y hacer referencia a un valor.
- Ese elemento reside en la memoria del ordenador.
- No habría ecuaciones y fórmulas sin variables.
- Tampoco habría programas sin variables.
- Una variable permite guardar información asociada a un nombre único.
- Es una característica transversal de todos los lenguajes de programación.
- En JavaScript no existe el tipado de variables, el tipado es dinámico.
- En la práctica, esto quiere decir que cuando se crea una variable no hay que asignarle un tipo.

Variables: Identificadores (I)

- Son los nombres de las variables.
- Reglas:
  - Pueden contener letras, números y guión bajo. No puede tener espacios.
  - El primer carácter tiene que ser una letra.
  - Se diferencia entre mayúsculas y minúsculas.
    - A los lenguajes que diferencian entre mayúsculas y minúsculas se les llama *case sensitive*.
  - No se pueden usar palabras reservadas, las palabras del propio lenguaje *(if, return)*.

Variables: Identificadores (II)

- Se pueden crear identificadores extensos, de hecho, muchas veces será mejor.
- Por ejemplo, para almacenar una edad la variable sería mejor llamarla *edad* que *e*.
- Para nombres compuestos podemos usar el guión bajo o la notación *camel case*. Ejemplos:
  - *edadPersona*
  - *edad_persona*

Uso de var

- En JavaScript tradicional, la palabra reservada *var* nos permitía declarar variables.
- *var* se mantiene por mantener la compatibilidad con el JavaScript tradicional.
- Razones por las que no usar *var*:
  - El alcance *(scope)* es confuso.
  - Se pueden emplear variables antes de declararlas.

Uso de let y const

- La palabra reservada *let* nos permite declarar variables.
- La palabra reservada *const* nos permite declarar variables cuyo valor no cambiará nunca en su ámbito de uso.
- La recomendación es usar *let* en aquellas variables que cambien su valor y *const* en las que no.
- Las reglas en los identificadores son las mismas que cuando se usa *var*.

Tipos de datos: Números

- Se utilizan para almacenar valores numéricos enteros o decimales.
- En este caso, el valor se asigna indicando directamente el número entero o decimal.
- Los números decimales utilizan el carácter del punto y no el de la coma para separar la parte entera de la parte decimal:

```
let iva = 21;          // variable tipo entero
let total = 234.65;    // variable tipo decimal.
```

Operadores de asignación

- El igual (=) es el operador clásico

Operador abreviado	Descripción	Significado
x += y	Asignación con suma	x = x + y
x -= y	Asignación con resta	x = x - y
x *= y	Asignación con multiplicación	x = x * y
x /= y	Asignación con división	x = x / y

Operadores aritméticos

- +, -, *, /: Suma, resta, producto y división.

Operador	Descripción	Ejemplo
% (Modulus)	Operador binario. Devuelve el resto de la división de dos operandos	12 % 5 devuelve 2.
++ (Incremento)	Operador unario. Suma uno al operando.	Si x es 3, x++ establece x a 4.
-- (Decremento)	Operador unario. Resta uno al operando.	Si x es 3, x-- establece x a 2.

El objeto Math

- El objeto *Math* es una biblioteca JavaScript de funciones de matemáticas que nos puede ayudar para ejecutar cálculos complejos.
- Es algo así como una calculadora científica.
- Si tecleamos *Math*. en un entorno JavaScript aparecerán todas las funciones.

Tipos de datos: Strings

- Se utilizan para almacenar caracteres, palabras y/o frases de texto.
- Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples para delimitar su comienzo y su final:

```
let mensaje = "Bienvenido a nuestro sitio web";
let nombreProducto = 'Producto ABC’;
let letraSeleccionada = 'c';
```

Tipos de datos: Strings complejos

- Para casos más complejos (comillas simples y dobles, tabulaciones, saltos de línea), sustituir carácter conflictivo por combinación. A esta técnica se le llama “mecanismo de escape”.

Si se quiere incluir...	Se debe incluir...
Una nueva línea	\n
Un tabulador	\t
Una barra inclinada	\\

Tipos de datos: Booleanos

- Una variable de tipo booleano almacena un tipo especial de valor que solamente puede tomar dos valores: **true** (verdadero) o **false** (falso).
- No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

```
let clienteRegistrado = false;
let ivaIncluido = true;
```

Tipos de datos: Valores nulos (I)

- Cuando accedemos a las variables, por circunstancias del programa a veces es necesario asegurarnos de que existan para no correr el riesgo de provocar un error que impida que se ejecute el resto del script.
- JavaScript tiene dos formas de avisarnos de que algo pasa con la variable u objeto al que deseamos acceder: las palabras **null** y **undefined**.

Tipos de datos: Valores nulos (II)

- Diferencias
  - *undefined* nos informa de que la variable no está tiene asignado ningún valor.
  - *null* es un contenido nulo que se le asigna.
- Una variable contendrá algún valor, si no es así será *null*, o si no *undefined*.

Tipos de datos: Objetos

- A menudo será necesario almacenar datos con una estructura más complicada que un simple dato que guardamos en una variable.
- Por ejemplo, si queremos guardar los datos de una persona (nombre, apellido, email, edad…), podemos crear una variable por cada dato o un objeto que agrupe todo.
- Los objetos son estructuras de datos que se definen como las variables.
- A la hora de asignarles el valor, este deberá ir entre {} y dentro, introduciremos las propiedades de este y sus valores separados del carácter :
- Para separar unas propiedades de otras utilizaremos la coma.
- Ejemplo:

```
const producto = {
    nombre: "Monitor 20 pulgadas",
    precio: 30,
    disponible: true
}
```

Arrays: ¿Qué son?

- Un array es un medio de guardar un conjunto de objetos o variables de la misma clase o tipo.
- Se accede a cada elemento individual del array mediante un número entero denominado índice.
- 0 es el índice del primer elemento y *n*-1 es el índice del último elemento, siendo *n* la dimensión del array.

Arrays: Declaración

- Para definir un array, se utilizan los caracteres [ ] para delimitar su comienzo y su final.
- Se utiliza el carácter , (coma) para separar sus elementos.
- Para acceder a los elementos del array se utilizan los corchetes teniendo en cuenta que van en base 0.

Funciones: Definición

- Una función es un conjunto de instrucciones que agrupamos bajo un determinado nombre.
- Cuando se referencia a estas instrucciones mediante el nombre de la función es cuando las instrucciones se ejecutan.
- Las funciones tienen dos características principales:
  - Permiten estructurar nuestro código.
  - Permiten repetir una serie de instrucciones de una forma sencilla.

Funciones: Ejecución

- Una cosa es crear una función y otra bien distinta es llamarla.
- Si creamos una función y no la llamamos nunca no se ejecutará nunca.
- Es como tener un coche siempre parado.
- Para usarlo debemos arrancarlo.
- Para que una función se dispare y se ejecute se debe llamar en algún momento del programa.

Funciones: Declaración

- Una función, en su declaración y desarrollo consta de tres partes:
  - **Nombre**
  - **Parámetros que recibe**: Complementan la función y a menudo son indispensables. Para calcular la edad la función deberá recibir de alguna manera la fecha de nacimiento. Los parámetros sirven para eso, para alimentar la función con ingredientes que necesita para poder ejecutarse.
  - **Cuerpo:** Es la funcionalidad.

Funciones: Retorno de datos

- De la misma manera que una función puede recibir datos de entrada gracias a los parámetros, también puede expulsar datos de salida mediante la palabra clave **return.**
- Estructura de una función en JavaScript:

```
function nombre_funcion (parámetro1, parámetro2, …)
{
        // Instrucciones. Con return expulsamos un valor
        return valor_salida
}
```

Funciones: Sintaxis general

- La declaración de una función se comienza con la palabra reservada **function.**
- A continuación damos un nombre a la función. Se aconseja identificar la función con un nombre significativo y asociado a su funcionalidad.
- Opcionalmente, se detallarán entre paréntesis los parámetros de entrada de la función.
- Si la función va a devolver un valor de salida, se expulsará con la palabra reservada **return.**

Bloque if

- La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura **if**.
- Se emplea para tomar decisiones en función de una condición.
- Se pueden anidar bloques if.
- La sintaxis básica es esta:

```
if(condicion) {
      // sentencias true
}
else {
      // sentencias false
}
```

Operadores de comparación o lógicos

- Devuelven un valor booleano.

Operador	Descripción	Ejemplo
<	Menor que. Compara dos valores devolviendo true si el primero es menor que el segundo y false en el caso contrario	1 < 2 devuelve true 2 < 1 devuelve false
< =	Menor o igual que. Compara dos valores devolviendo true si el primero es menor o igual que el segundo y false en el caso contrario	1 <= 2 devuelve true 2 <= 1 devuelve false 2 <= 2 devuelve true
>	Mayor que. Compara dos valores devolviendo true si el primero es mayor que el segundo y false en el caso contrario	1 > 2 devuelve false 2 > 1 devuelve true
> =	Mayor o igual que. Compara dos valores devolviendo true si el primero es mayor o igual que el segundo y false en el caso contrario	1 >= 2 devuelve false 2 >= 1 devuelve true 2 >= 2 devuelve true
= =	Igual a. Compara dos valores devolviendo true si son iguales y false en caso contrario	1 == 1 devuelve false 2 == 1 devuelve true
= = =	Estrictamente igual. Compara dos valores devolviendo true si son iguales y tienen el mismo tipo	1 === 1 devuelve true 1 === '1' devuelve false
!=	Distinto a. Compara dos valores devolviendo true si son distintos y false en caso contrario	1 != 1 devuelve false 2 != 1 devuelve true

Bloque SWITCH

- Se utiliza para evaluar múltiples condiciones y tomar bifurcaciones en función al valor que tome una determinada variable.
- Su uso también es igual que en otros lenguajes.
- Se usa la instrucción **break** para romper el bloque y salirse de él.
- Para evaluar una última condición se puede usar **default**.

Operador AND

- La operación lógica AND obtiene su resultado combinando dos o más valores booleanos.
- El operador se indica mediante los caracteres && y su resultado solamente es true si todos los casos son true.

Operador OR

- La operación lógica OR también combina dos o más valores booleanos.
- El operador se indica mediante los caracteres|| y su resultado es true si alguno de los dos casos es true.

Estructura for

- Permite construir un bloque de instrucciones iterativo.
- Repite las instrucciones del bloque un número determinado de veces.
- Después de cada repetición actualiza unas variables. Estructura:

```
for(inicializacion; condicion; actualizacion) {
   ...
}
```

Sentencia break

- Esta sentencia va a permitir alterar el comportamiento normal de los bucles.
- Permite salirnos bruscamente de ellos aun cumpliéndose las condiciones para que el bucle se siga ejecutando.

Sentencia continue

- Esta sentencia indica que se continúe con el bucle pero desde el inicio del mismo.
- Todo lo que aparezca debajo de *continue* es ignorado para esa iteración.

Estructura while

- Permiten repetir sentencias de código mientras se cumpla una condición. Sintaxis:

```
while(condición) {
   ...
}

do {
   ...
} while(condición);
```

- Ambas estructuras son iguales salvo que la segunda asegura la entrada al bucle por lo menos la primera vez.
- Después evaluará si repite la iteración o no.

DOM

- El DOM es una jerarquía de objetos que describen los elementos de la página web que está mostrando el navegador.
- El DOM permite a los programadores web acceder y manipular las páginas HTML.
- Para poder utilizar las capacidades del DOM es necesario transformar en algo visual la sucesión de caracteres que conforman el código HTML.
- DOM transforma todos los documentos HTML en un conjunto de elementos llamados nodos.
- Estos interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos.

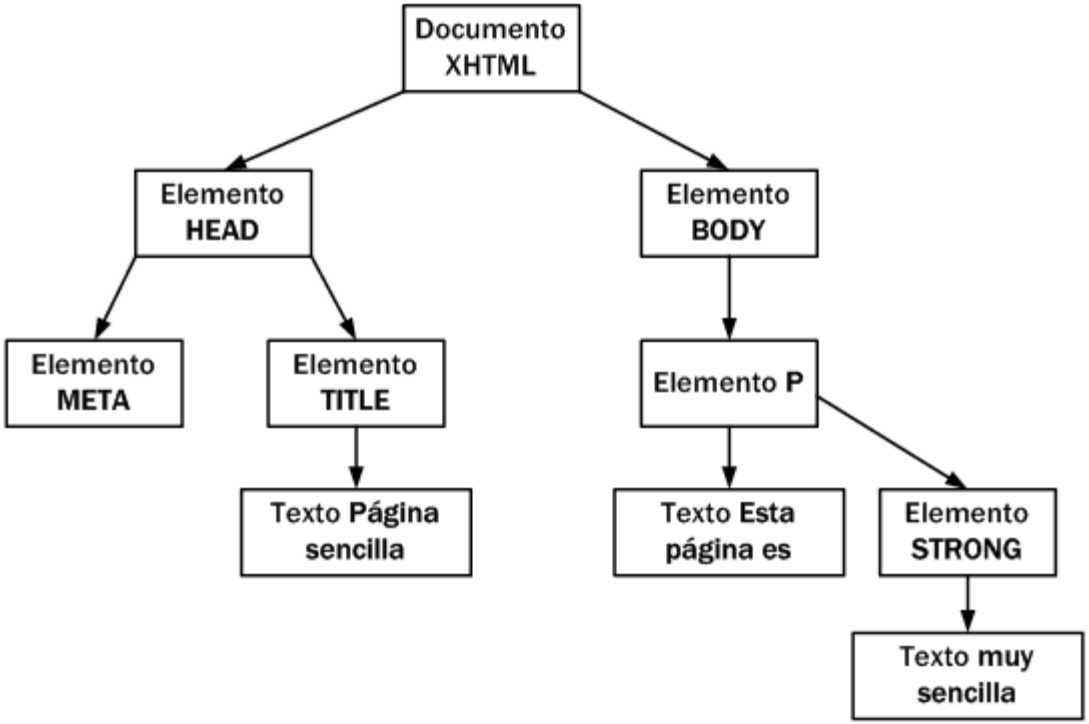
DOM: Ejemplo (I)

- Ejemplo:

```
<html>
<head>
<title>Página sencilla</title>
<meta charset="UTF-8"/>
</head>

<body>
<p>Esta página es <strong>muy
sencilla</strong></p>
</body>
</html>
```

DOM: Ejemplo (II)



DOM: Estructura

- La raíz del árbol de nodos de cualquier página HTML siempre es la misma:
  - Un nodo de tipo especial denominado **window** (Ventana) que hace referencia al navegador con otro nodo hijo llamado:
  - **document** (Documento o página web).
- A partir de ese nodo raíz, cada etiqueta HTML se transforma en un nodo de tipo "Elemento".
- La conversión de etiquetas en nodos se realiza de forma jerárquica.

Acceso a los nodos del DOM: **getElementsByTagName(etiqueta)**

- Esta función obtiene un array todos los elementos de la etiqueta que se especifica.
- El siguiente ejemplo muestra cómo obtener todos los párrafos de una página HTML:
- var parrafos =document.getElementsByTagName("p");
- Se puede obtener el primer párrafo de la página de la siguiente manera:

```
const parrafos
=document.getElementsByTagName('p');
const primerParrafo = parrafos[0];
```

Acceso a los nodos del DOM
getElementsByClassName(clase)

- Esta función es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo **class** sea igual al parámetro proporcionado.
- Puede ser muy útil cuando queremos hacer una captura de elementos de la misma clase.

Acceso a los nodos del DOM
getElementById(id)

- Permite 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.

Acceso a los nodos del DOM
querySelector(selector css) y querySelectorAll(selector css)

- **querySelector(selector css):** Selecciona un único elemento basándose en el selector css que especificamos.
- **querySelectorAll(selector css):** Selecciona todos los elementos basándose en el selector css que especificamos.

Creación y eliminación de nodos (I)

- Crear y añadir a la página un nuevo elemento HTML sencillo consta de tres pasos diferentes:
  - Creación de un nodo de tipo **Element** que represente al elemento.
  - Darle características.
  - **Añadir el nodo Element a la página** en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento dentro del DOM.

Creación y eliminación de nodos (II)

- **createElement(etiqueta):** Crea un nodo de tipo Element que representa al elemento HTML cuya etiqueta se pasa como parámetro.
- **nodoPadre.appendChild(nodoHijo):** Añade un nodo como hijo de otro nodo.

Creación y eliminación de nodos (III)

- La eliminación es más sencilla.
- Podemos invocar al método **remove** de un nodo.
- Otra manera es mediante la función **removeChild(elemento)**.
- Esta función requiere como parámetro el nodo que se va a eliminar y debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar.

Eventos (I)

- Hasta ahora, todas las aplicaciones y scripts que hemos visto tienen algo en común: Se ejecutan desde la primera instrucción hasta la última de forma secuencial.
- Las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de programación basada en eventos.
- JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas web.

Eventos (II)

- 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 constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.
- JavaScript permite asignar una función a cada uno de los eventos.

Eventos (III)

- El siguiente ejemplo muestra cómo definir un evento click en un botón:

```
<button onclick="calcular()">Haz clic para calcular</button>
```

¿Qué es POO? (I)

- No es un lenguaje de programación sino una forma de programar, una metodología.
- Intenta llevar al código lo mismo que encontramos en el mundo real.
- Cuando miramos a nuestro alrededor, ¿qué vemos?
- Vemos OBJETOS.

¿Qué es POO? (II)

- OBJETOS.
  - Podemos distinguir un objeto por que pertenece a una **clase**.
  - Distinguimos un perro de un coche porque pertenecen a clases diferentes.
  - Distinguimos un televisor de otro porque aunque pertenezcan a la misma clase, cada objeto es diferente.
- Esta es el modelo que intenta seguir la POO para estructurar un sistema.

Características de la POO (I)

- **Abstracción:**
  - Es algo conceptual.
  - Define nuevos tipos de entidades en el mundo real.
  - Depende de la perspectiva del observador.
  - Se centra en la visión externa de un objeto, Ej.: Pastor, pekinés o bulldog son derivados de la idea abstracta "perro".

Características de la POO (II)

- **Jerarquía:**
  - Las abstracciones forman una jerarquía.
  - Seccionamos el código en partes más pequeñas que al unirlas hacen el trabajo.
  - Ej.: Máquinas de locomoción →de tierra →sin motor→ con pedales → para niños → triciclo.

Características de la POO (III)

- **Encapsulación y ocultamiento:**
  - La encapsulación se encarga de mantener ocultos los procesos internos que se necesitan para hacer lo que sea que haga. → El programador accede sólo a lo que necesita.

Características de la POO (IV)

- **Encapsulación y ocultamiento:**
  - Un objeto se comunica con el resto del mundo mediante su **interfaz** → la lista de métodos y propiedades que hace público.
  - Ventaja: Lo que hace el usuario puede ser controlado internamente, incluso los errores, evitando que todo colapse.

Características de la POO (V)

- **Herencia:**
  - Capacidad que tiene una clase de derivar las propiedades y métodos de otra.
  - Decimos que una gallina es un ave → las gallinas tienen características comunes con otras aves (pico, plumas…) → la gallina hereda las características comunes de aves. → Además un ave es un animal → comparte características comunes con el caballo, el perro, el hombre.

Características de la POO (VI)

- **Herencia:**
  - Ventajas: Evita tener que escribir el mismo código una y otra vez ya que al definir una categoría (que llamaremos clase) que pertenece a otra, automáticamente atribuimos las características de la primera sin tener que definirlas de nuevo.
  - Ejemplo: Controles Formularios Windows →Cuadro de texto → Cuadro de texto con máscara

Clase (I)

- Sabemos que “ave” se refiere a algo que cumple unas determinadas características, se comporta de una forma concreta → tiene plumas, pico, dos patas… → “ave” es una palabra que nos permite clasificar las cosas.
- “ave” identifica a un grupo, no a un ave en concreto.
- Es un concepto, una abstracción.

Clase (II)

- La clasificación (crear clases) es la tarea de agrupar cosas según sus características y sus capacidades.
- **Cada objeto debe pertenecer a una clase específica**, de acuerdo a sus características y si esa clase no existe, se crea.

Clase (III)

- Por lo tanto, una clase es una serie de código que define a todos los elementos relacionados → es una plantilla, un molde a partir de la que se crean los objetos.
- La clase posee todas las variables y funciones que deben tener un objeto → el objeto tendrá valores propios en esas variables cuando sea creado.
- Un objeto no es una clase, sino el **resultado de crear un ejemplar de esa clase**.

Clase (IV)

- JavaScript tiene muchas clases preconstruidas que utilizamos para desarrollar aplicaciones.
- Formato de una clase en JavaScript:

```
class NombreClase {
    constructor
    propiedades
    métodos
}
```

- La palabra clave **class** siempre encabeza la definición de una clase.

Objeto

- Un objeto es cualquier cosa del mundo real: una bici, un perro, un televisor, una excursión…
- Todos los objetos tienen 2 características:
  - Propiedades/características
  - Comportamiento
- Ej.: perro → nombre, color, raza → ladrar, buscar, menearCola.
- En nuestras clases almacenaremos las propiedades de un objeto como propiedades y los comportamientos como métodos.

Ejemplares/Objetos de una clase (I)

- Una clase es una acción declarativa → no ocurre nada hasta que no creamos su objeto.
- A la creación de un ejemplar de una clase se le denomina instanciación. Al objeto, instancia.
- Ej.: creamos un objeto de la clase TV

```
TV miTV = new TV()
```

- El operador new seguido del nombre de la clase, crea un objeto de dicha clase en una posición de memoria a la que accederemos con el nombre miTV.

Ejemplares/Objetos de una clase (II)

- Ahora puedo usar libremente lo que me permita la interfaz de miTV.
- Ejemplos:

```
miTV.Encender()
miTV.CambiarCadena(2)
miTV.Apagar()
```

Atributos / variables

- Forman la estructura interna de los objetos de una clase. Son los que hacen a un objeto diferente de otro.
- Se declaran exactamente igual que cualquier otra variable pudiendo incluso inicializarse con un valor.

Constructores (I)

- Son métodos que se llaman automáticamente al crear un objeto de una clase.
- Su misión principal es iniciar nuevos objetos de una clase.
- El método constructor no se puede llamar, se autoejecuta al crear un objeto con la palabra reservada **new**.

Métodos (I)

- Un método es un conjunto de declaraciones asociadas a un nombre.
- Se ejecutarán todas cuando se llame al método.
- Forman lo que se denomina interfaz o medio de acceso a la estructura interna de las clases.
- Definen las operaciones que pueden realizarse con las propiedades.

JSON

- JSON (acrónimo de JavaScript Object Notation), es un formato de texto sencillo para el intercambio de datos.
- Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.
- Hoy en día es el lenguaje de intercambio de datos estándar cliente/servidor.

JSON: Sintaxis

- Los tipos de datos disponibles en JSON son:
  - **Números**: Se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos. Ejemplo: 123.456
  - **Cadenas**: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla. Ejemplo: "Hola"
  - **Booleanos**: Representan valores booleanos y pueden tener dos valores: true y false
  - **null**: Representan el valor nulo.
  - **Array**: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo ["juan","pedro","jacinto"]
  - **Objetos**: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena entre comillas dobles. El valor puede ser de cualquier tipo.

JSON: Ejemplo


```
{
  "departamento":8,
  "nombredepto":"Ventas",
  "director": "Juan Rodríguez",
  "empleados":[
    {
      "nombre":"Pedro",
      "apellido":"Fernández"
    },{
      "nombre":"Jacinto",
      "apellido":"Benavente"
    }
  ]
}
```

Depuración de errores

- Los errores son parte del trabajo de programar.
- El trabajo de depuración es importante.
- También disponer de herramientas.
- Tipos de errores.
  - Sintácticos
  - De ejecución o lógicos

Errores sintácticos (I)

- Se generan por infringir las normas de escritura de un lenguaje.
- Suelen deberse a olvidos o a desconocimiento.
- Ejemplos:
  - Falta o mal uso de elementos separadores (comas, puntos y comas, dos puntos, etc.).
  - Palabras mal escritas (por ejemplo en JavaScript IF o EF en vez de if).

Errores sintácticos (II)

- Suelen ser errores obvios y fáciles de detectar.
- La mayoría de los lenguajes tienen herramientas de ayuda para solucionarlos.
- No es posible la ejecución de un programa con errores de sintaxis.

Errores de ejecución (I)

- Son todos aquellos derivados de un mal diseño de los algoritmos.
- Son los que más grandes quebraderos de cabeza originan a los programadores.
- Los más difíciles de evitar y los más difíciles de detectar.

Buenas prácticas para evitar errores (I)

- **Tabular el código:** Una tabulación precisa clarifica de una manera excelente el código. En JavaScript nos va a permitir distinguir las funciones del resto de código o ver dónde comienzan y terminan las estructuras.
- **Escribir comentarios aclaratorios:** Se debe evitar comentar lo más obvio pero tampoco hay que ser parcos en la documentación del código. Nos ayudará en su mantenimiento posterior.

Buenas prácticas para evitar errores (II)

- **Separar el código en la medida de lo posible:** Mezclar código HTML, CSS y JavaScript en un mismo documento, aunque posible, no es recomendable.
- Separar el código en archivos independientes según lenguajes y tecnologías va a aportar mucho en cuanto a la claridad y posterior mantenimiento del código.

Bloque try/catch.

- Habrá muchas veces que los errores provengan de otras fuentes externas.
- Por ejemplo, puede que nuestro código ejecute una función que conecte con una base de datos remota. ¿Qué pasaría si el servidor de base de datos remoto estuviera caído? Pues que se originaría un error sin tener nosotros la culpa.
- No obstante, si es labor nuestra gestionar este tipo de situaciones potencialmente críticas.
- Solución: Bloque **try/catch**