PROGRAMACIÓN WEB EN EL ENTORNO CLIENTE

MF0491_3

Gobierno de Canarias
Servicio Canario de Empleo

FUERTEVENTURA 2000
Centro de Formación Profesional

PIEC
PLAN INTEGRAL DE EMPLEO DE CANARIAS

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL
SEPE

Realizado por Manuel Macías

Qué es la lógica de programación

La lógica de programación consiste en la organización y planificación coherente de las instrucciones necesarias para ejecutar con éxito un programa.

Dentro de ello hay una serie de patrones que se repiten en todos los lenguajes, como el uso de variables, métodos o funciones, condicionales y bucles.

Qué es un ordinograma

Se trata de un estilo específico de diagramas de flujo, que muestra la secuencia lógica y detallada de las operaciones que necesitamos para la realización de un programa, paso a paso y con sus distintos caminos, tomas de decisiones, repeticiones, etc.

Símbolos utilizados

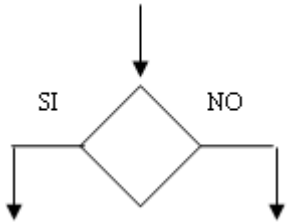
Descripción: Para inicio/fin o para una parada indeterminada

Descripción: Símbolo de entrada/salida genérico

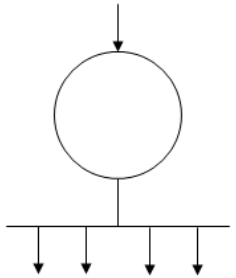
Descripción: Representa una operación o proceso general con datos de memoria.

Símbolos utilizados

Descripción: Símbolo de subprograma o subrutina. Se utiliza para realizar una llamada a un modulo del programa.



Descripción: Símbolo de decisión para realizar una pregunta con dos posibles respuestas. Es lo que llamamos símbolo de selección simple.

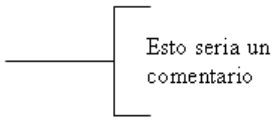


Descripción: Símbolo de selección múltiple

Símbolos utilizados

Descripción: Símbolo de bucle definido.

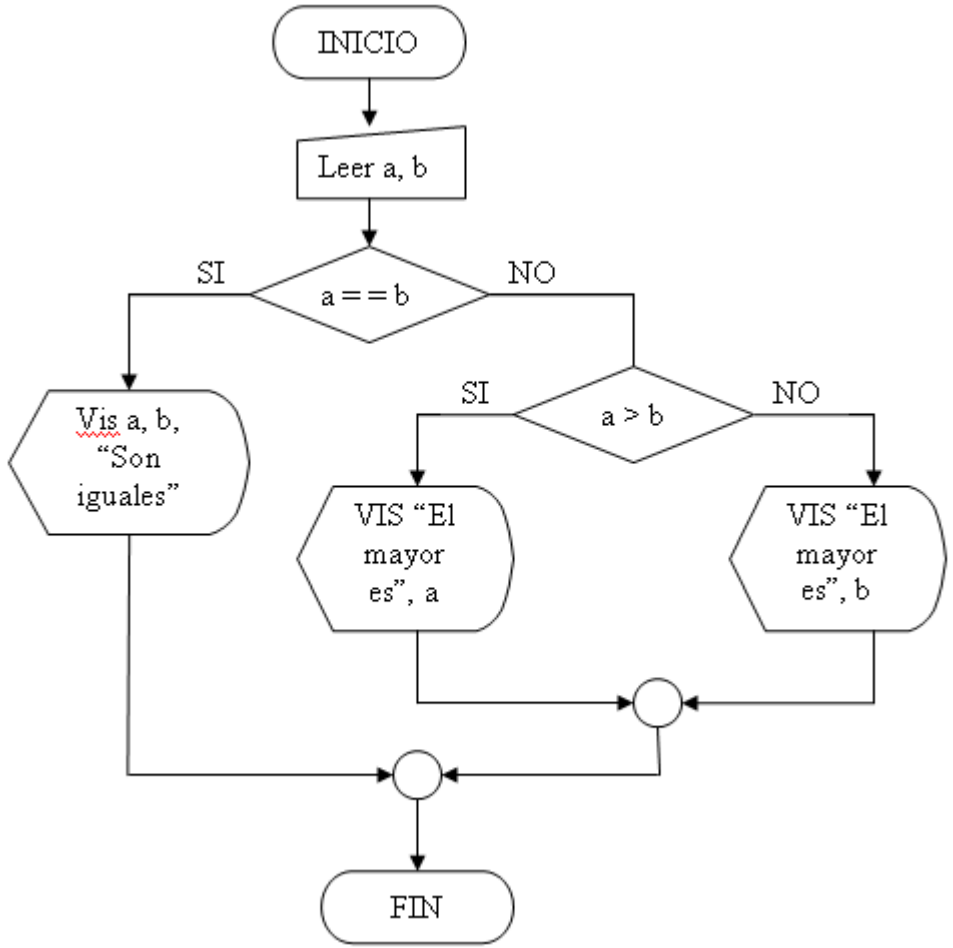
Descripción: Conector. Se utiliza para agrupar varias líneas de flujo que salen del mismo origen.



Descripción: Símbolo para poner comentarios

Reglas a la hora de hacer ordinogramas

- Todos los símbolos utilizados deben estar unidos por líneas de flujo.
- No se pueden cruzar las líneas de flujo
- A un símbolo de proceso pueden llegarle varias líneas de flujo pero solo puede salir una de él.
- Al símbolo de inicio no puede llegarle ninguna línea de flujo
- De un símbolo de fin no puede salir ninguna línea de flujo pero si le pueden llegar varias.



Ejemplo de ordinograma

¿Qué Es el Pseudocódigo?

El pseudocódigo es una forma de representar código, como algoritmos, funciones y otros procesos, utilizando una combinación de lenguaje natural y elementos similares al lenguaje de programación.

Se llama «pseudocódigo» porque no es realmente ejecutable. En cambio, es una forma de que los humanos comprendan y planifiquen la lógica de la programación — describir los pasos de un programa de forma que sea fácil de entender para los humanos, sin dejar de ser lo suficientemente detallado como para convertirse rápidamente en un lenguaje de programación específico.

Ejemplo de Pseudocódigo.

Dadas dos variables numéricas A y B, que el usuario debe teclear, se pide realizar un algoritmo que intercambie los valores de ambas variables y muestre cuanto valen al final las dos variables:

```
InicioProceso
   Escribir "Introduce el valor de A"
   Leer A
   Escribir "Introduce el valor de B"
   Leer B
   C<-A
   A<-B
   B<-C
   Escribir "A vale " A " y B vale " B
FinProceso
```

Programa:

En programación se suele llamar «programa» al conjunto total de código que se desarrolla.

En Javascript, quizás el término más utilizado es aplicación web (cuando es un desarrollo con mucha cantidad de Javascript). También se suelen generalizar utilizando términos como «script» o «código Javascript».

Algoritmo:

Un algoritmo es un conjunto de pasos conocidos, en un determinado orden, para conseguir realizar una tarea satisfactoriamente y lograr un objetivo.

Comentarios:

Los comentarios en nuestro código son fragmentos de texto o anotaciones que el navegador ignora y no repercuten en el programa. Sirven para dejar por escrito detalles importantes para el programador. De esta forma cuando volvamos al código, nos será más rápido comprenderlo.

Es una buena costumbre comentar en la medida de lo posible nuestro código.

Indentación:

Se llama indentar a la acción de colocar espacios o tabuladores antes del código, para indicar si nos encontramos dentro de un if, de un bucle, etc...

Variables:

Es el nombre genérico que se le da a pequeños espacios de memoria donde guardas una información determinada, de forma muy similar a las incógnitas en matemáticas. Un programa puede tener muchas variables, y cada una de ellas tendrá un nombre, un valor y un tipo de dato.

Constantes:

Es el mismo concepto de una variable, salvo que en este caso, la información que contiene es siempre la misma (no puede variar).

Funciones:

Cuando comenzamos a programar, nuestro código se va haciendo cada vez más y más grande, por lo que hay que buscar formas de organizarlo y mantenerlo lo más simple posible. Las funciones son agrupaciones de código que, entre otras cosas, evitan que tengamos que escribir varias veces lo mismo en nuestro código. Una función contendrá una o mas acciones a realizar y cada vez que ejecutemos una función, se realizarán todas ellas.

Parámetros:

Es el nombre que reciben las variables que se le pasan a las funciones. Muchas veces también se les denomina argumentos.

Bucles:

Cuando estamos programando, muchas veces necesitaremos realizar tareas repetitivas. Una de las ventajas de la programación es que permite automatizar acciones y no es necesario hacerlas varias veces. Los bucles permiten indicar el número de veces que se repetirá una acción. De esta forma, sólo la escribimos una vez en nuestro código, y simplemente indicamos el número de veces que queremos que se repita.

Iteración:

Cuando el programa está en un bucle repitiendo varias veces la misma tarea, cada una de esas repeticiones se denomina iteración.

Librería:

Muchas veces, desarrollamos código que resuelve tareas o problemas que, posteriormente, querremos reutilizar en otros programas. Cuando eso ocurre, en Javascript se suele empaquetar el código en lo que se llaman librerías, que no es más que código listo para que otros programadores puedan utilizarlo fácilmente en sus programas y beneficiarse de las tareas que resuelven de forma muy sencilla.

Elementos del lenguaje de guion:

- Variables (Numéricas, Caracteres, Cadena de Caracteres, Booleanos)
  - Variables Globales
  - Variables Locales

Elementos del lenguaje de guion:

- Operaciones
  - Aritméticas
  - Lógicas
  - Binarias
  - Otras

Elementos del lenguaje de guion:

- Comparaciones
  - Igual ‘==‘).
  - Distinto ‘!=‘).
  - Mayor ‘>’).
  - Menor’<‘).
  - Mayor o igual ‘>=‘).
  - Menor o igual ‘<=‘).

Elementos del lenguaje de guion:

- Asignaciones
  - nombre_variable = valor;

No confundir Comparación (==) y asignación (=).

Objetos del lenguaje de guion:

- Métodos (función asociada a un objeto)
- Eventos (acción del usuario en nuestra web a la que tenemos que dar respuesta.
- Atributos (definen datos particulares sobre el estado de un objeto)
- Funciones (Es un conjunto de instrucciones que agrupamos bajo un determinado nombre)
  - Definición o nombre de la función
  - Parámetros que recibe la función
  - Cuerpo principal de la función

Objetos del lenguaje de guion:

```
Nombre función sumar (a,b) {
        Return a+b;
}
```

```
function nombre(param,param, ...param) {
   instrucciones
}
```

Ejecución de un script.

Los scripts pueden ejecutarse o cuando la página se está cargando o cuando se produce un determinado evento.

Declaración de variables

Para poder trabajar con variables , previamente tenemos que declararlas y darles un identificador para poder referenciarlas en nuestro programa.

Las operaciones con variables las podemos usar para:

- Mostrar datos.
- Pedir datos.
- Enviar datos.
- Recibir datos.
- En el uso de expresiones.
- En funciones.
- En llamadas a las funciones.
- En objetos.

Declarando una variable en JavaScript

```
var variable = 2;
```

Usando el contenido de una variable

```
var variable = 2;
console.log(variable);
```

Asignando un nuevo valor a la variable

```
var variable = 2;
 variable = 3;
console.log(variable);
```

Asignando el valor de una variable a otra variable

```
var variable = 2;
var variable2 = variable;
console.log(variable);
```

Realizando una operación con el valor almacenado en una variable

```
var variable = 2;
 variable = variable + 1;  // 3
console.log(variable);
```

Realizando operaciones complejas

```
var variable = 2;
 variable = variable * 25 + 3;  // 53
console.log(variable);
```

Controlando la precedencia en la operación

```
var variable = 2;
 variable = variable * (25 + 3);  // 56
console.log(variable);
```

Incrementando el valor de una variable

```
var variable = 0;
 variable ++;
console.log(variable);  // 1
```

Incrementando el valor de una variable en un valor específico

```
var variable = 0;
 variable += 5;
console.log(variable);  // 5
```

Asignando una cadena de caracteres a una variable

```
var variable = "Hola Mundo!";
console.log(variable);
```

Concatenando texto

```
var variable = "Mi nombre es ";
 variable = variable + "Juan";
console.log(variable);
```

Agregando texto al comienzo del valor

```
var variable = "Juan";
 variable = "Mi nombre es " + variable;
console.log(variable);
```

Concatenando texto con números

```
var variable = "El número es " + 3;
console.log(variable);
```

Generando nuevas líneas de texto

```
var variable = "Felicidad no es hacer lo que uno quiere\r\n";
 variable = variable + "sino querer lo que uno hace."
console.log(variable);
```

Declarando una variable Booleana

```
var variable = true;
console.log(variable);
```

Creando arrays

```
var array = ["rojo", "verde", "azul"];
alert(array[0]);  // "rojo"
```

Mostrando los valores del array

```
var array = ["rojo", "verde", "azul"];
alert(array);  // "rojo,verde,azul"
```

Almacenando diferentes tipos de valores

```
var array = ["rojo", 32, "HTML5 es genial!"];
alert(array[1]);
```

Declarando valores indefinidos

```
var array = ["rojo", undefined, 32];
alert(array[1]);
```

Declarando el valor null

```
var array = ["rojo", 32, null];
alert(array[2]);
```

Declarando valores indefinidos

```
var array = ["rojo", undefined, 32];
alert(array[1]);
```

Declarando el valor null

```
var array = ["rojo", 32, null];
alert(array[2]);
```

Igualdad Estricta (===)

El operador de estricta igualdad (===) revisa si dos operandos son iguales y produce un resultado Booleano.

A diferencia del operador de igualdad regular (==), el operador de estricta igualdad siempre considera que los operandos de distinto tipo de valor son diferentes y nunca similares.

Tipos de datos

Number: El tipo number representa tanto números enteros como de punto/coma flotante.
Además de los números comunes, existen los llamados "valores numéricos especiales" que también pertenecen a este tipo de datos: Infinity, -Infinity y NaN.

Tipos de datos

Number

Infinity representa el Infinito matemático ∞. Es un valor especial que es mayor que cualquier número.

NaN representa un error de cálculo. Es el resultado de una operación matemática incorrecta o indefinida.

Tipos de datos

BigInt

En JavaScript, el tipo “number” no puede representar de forma segura valores enteros mayores que (253-1) (eso es 9007199254740991), o menor que -(253-1) para negativos.

Para la mayoría de los propósitos, el rango ±(253-1) es suficiente, pero a veces necesitamos números realmente grandes; por ejemplo, para criptografía o marcas de tiempo de precisión de microsegundos.

BigInt se agregó recientemente al lenguaje para representar enteros de longitud arbitraria.

Un valor **BigInt** se crea agregando **n** al final de un entero

Tipos de datos

String

Un string en JavaScript es una cadena de caracteres y debe colocarse entre comillas.

```
let str = "Hola";
let str2 = 'Las comillas simples también están bien';
let phrase = `se puede incrustar otro ${str}`;
```

Tipos de datos

String

En JavaScript, hay 3 tipos de comillas.

Comillas dobles: "Hola".
Comillas simples: 'Hola'.
Backticks (comillas invertidas): `Hola`.

Las comillas dobles y simples son comillas “sencillas” (es decir, funcionan igual). No hay diferencia entre ellas en JavaScript.
Los backticks son comillas de “funcionalidad extendida”. Nos permiten incrustar variables y expresiones en una cadena de caracteres encerrándolas en ${...},

Tipos de datos

Boolean (tipo lógico)
El tipo boolean tiene sólo dos valores posibles: true y false.

Este tipo se utiliza comúnmente para almacenar valores de sí/no: true significa “sí, correcto, verdadero”, y false significa “no, incorrecto, falso”.

Tipos de datos

El valor “null” (nulo)

El valor especial null no pertenece a ninguno de los tipos descritos anteriormente.

Forma un tipo propio separado que contiene sólo el valor null:

```
let age = null;
```

En JavaScript, null no es una “referencia a un objeto inexistente” o un “puntero nulo” como en otros lenguajes.
Es sólo un valor especial que representa “nada”, “vacío” o “valor desconocido”.
El código anterior indica que el valor de age es desconocido o está vacío por alguna razón.

Tipos de datos

El valor “undefined” (indefinido)
El valor especial undefined también se distingue. Hace un tipo propio, igual que null.

El significado de undefined es “valor no asignado”.

Si una variable es declarada, pero no asignada, entonces su valor es undefined:
let age;

alert(age); // muestra "undefined"

Tipos de datos

Object y Symbol
El tipo object (objeto) es especial.

Todos los demás tipos se llaman “primitivos” porque sus valores pueden contener una sola cosa (ya sea una cadena, un número o lo que sea). Por el contrario, los objetos se utilizan para almacenar colecciones de datos y entidades más complejas.
El tipo symbol (símbolo) se utiliza para crear identificadores únicos para los objetos.

Tipos de datos

El operador typeof

El operador typeof devuelve el tipo de dato del operando. Es útil cuando queremos procesar valores de diferentes tipos de forma diferente o simplemente queremos hacer una comprobación rápida.

La llamada a typeof x devuelve una cadena con el nombre del tipo:

Tipos de datos

El operador typeof

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"  (1)
typeof null // "object"  (2)
typeof alert // "function"  (3)
```

Tipos de datos

El operador typeof

1. Math es un objeto incorporado que proporciona operaciones matemáticas.

2. El resultado de typeof null es "object". Esto está oficialmente reconocido como un error de comportamiento de typeof que proviene de los primeros días de JavaScript y se mantiene por compatibilidad. Definitivamente null no es un objeto. Es un valor especial con un tipo propio separado.

3. El resultado de typeof alert es "function" porque alert es una función.

Arrays

Los objetos te permiten almacenar colecciones de datos a través de nombres.

Pero a menudo necesitamos una colección ordenada, donde tenemos un 1ro, un 2do, un 3er elemento y así sucesivamente.

Por ejemplo, necesitamos almacenar una lista de algo: usuarios, bienes, elementos HTML, etc.

No es conveniente usar objetos aquí, porque no proveen métodos para manejar el orden de los elementos. No podemos insertar una nueva propiedad “entre” los existentes. Los objetos no están hechos para eso.

Existe una estructura llamada Array (llamada en español arreglo o matriz/vector) para almacenar colecciones ordenadas.

Creando arrays

```
var miarray = ["rojo", "verde", "azul"];

alert(miarray);  // "rojo,verde,azul"

    alert(miarray[0]);  // "rojo"
```

Creando arrays

```
let fruits = ["Apple", "Orange", "Plum"];

alert( fruits[0] ); // Apple
alert( fruits[1] ); // Orange
alert( fruits[2] ); // Plum
```

Métodos pop/push, shift/unshift

Una **cola** es uno de los usos más comunes de un array. En ciencias de la computación, significa una colección ordenada de elementos que soportan dos operaciones:

push: inserta un elemento al final.
shift: obtiene el elemento del principio, avanzando la cola, y así el segundo elemento se vuelve primero.

Métodos pop/push.

Hay otro caso de uso para los arrays, la estructura de datos llamada **pila**. Esta soporta dos operaciones:

push: agrega un elemento al final.
pop: toma un elemento desde el final.

Entonces los elementos nuevos son agregados o tomados siempre desde el “final”.

Métodos pop/push

Para las pilas, la última introducida es la primera en ser recibida, en inglés esto es llamado principio LIFO (Last-In-First-Out, última en entrar primera en salir).
Para las colas, tenemos FIFO (First-In-First-Out primera en entrar, primera en salir).

Los arrays en JavaScript pueden trabajar como colas o pilas. Ellos permiten agregar o quitar elementos al o del principio o al o del final.

Métodos shift/unshift

shift

Extrae el primer elemento del array y lo devuelve:

```
let fruits = ["Apple", "Orange", "Pear"];

alert( fruits.shift() ); // quita Apple y lo muestra en un alert

alert( fruits ); // Orange, Pear
```

Métodos shift/unshift

Unshift

Agrega el elemento al principio del array:

```
let fruits = ["Orange", "Pear"];

fruits.unshift('Apple');

alert( fruits ); // Apple, Orange, Pear
```

Métodos shift/unshift

Unshift

Agrega el elemento al principio del array:

```
let fruits = ["Orange", "Pear"];

fruits.unshift('Apple');

alert( fruits ); // Apple, Orange, Pear
```

Splice

Esto modifica arr comenzando en el índice start: remueve la cantidad deleteCount de elementos y luego inserta elem1, ..., elemN en su lugar. Lo que devuelve es un array de los elementos borrados.

```
let arr = ["Yo", "estudio", "JavaScript", "ahora", "mismo"];
arr.splice(0, 3, "a", "bailar"); // borra los primeros 3 elementos y los reemplaza con otros

alert( arr ) // ahora ["a", "bailar", "ahora", "mismo"]
```

Funciones

Las funciones se declaran usando la palabra clave (**function**), el nombre seguido de paréntesis, y el código entre llaves.

Para llamar a la función (ejecutarla), tenemos que declarar su nombre con un par de paréntesis al final.

```
function mostrarMensaje(){
alert("Soy una función");
}
mostrarMensaje();
```

Objetos

Son estructuras de información capaces de contener variables (llamadas propiedades), así como funciones (llamadas métodos). Son como programas independientes que se comunican entre sí para realizar tareas comunes.

Objetos

El objeto se declara como cualquier otra variable usando la palabra clave var, y las propiedades y métodos que definen el objeto se declaran entre llaves usando dos puntos después del nombre y una coma para separar cada declaración.

```
var miojeto ={
Nombre: “Juan”,
Edad: 30
};
```

Objetos

A diferencia de las variables, no podemos acceder a los valores de las propiedades de un objeto usando solo sus nombres; también tenemos que especificar el nombre del objeto al que pertenecen usando notación de puntos o corchetes.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30
};
var mensaje = "Mi nombre es " + miobjeto.nombre + "\r\n";
mensaje += "Tengo " + miobjeto["edad"] + " años";
alert(mensaje);
```

Métodos

Las funciones dentro los objetos se llaman métodos, tienen la misma sintaxis que las propiedades: requieren dos puntos después del nombre y una coma para separar cada declaración, pero en lugar de valores, debemos asignarles funciones anónimas.

Métodos

```
var miobjeto = {
  nombre: "Juan",
  edad: 30,
  mostrardatos: function() {
    var mensaje = "Nombre: " + miobjeto.nombre + "\r\n";
    mensaje += "Edad: " + miobjeto.edad;
    alert(mensaje);
  },
  cambiarnombre: function(nombrenuevo) {
    miobjeto.nombre = nombrenuevo;
  }
};
miobjeto.mostrardatos();  // "Nombre: Juan Edad: 30"
miobjeto.cambiarnombre("José");
miobjeto.mostrardatos();  // "Nombre: José Edad: 30"
```

Objetos:

- Objetos estándar.
- Objetos Strings.
- Objetos Array.
- Objetos Date.
- Objeto Math.
- Objeto Window.
- Objeto Document.
- Objetos Element.
- Objetos Event.

Las sentencias JavaScript suelen comenzar con una **palabra clave** para identificar la acción JavaScript que se va a realizar.

Palabra clave	Descripción
var	Declara una variable
let	Declara una variable de bloque
const	Declara una constante de bloque
if	Marca un bloque de sentencias a ejecutar bajo una condición
switch	Marca un bloque de sentencias a ejecutar en diferentes casos
for	Marca un bloque de sentencias a ejecutar en un bucle
function	Declara una función
return	Sale de una función
try	Implementa la gestión de errores en un bloque de sentencias

Los operadores de asignación asignan valores a variables JavaScript. Los operadores de asignación asignan valores a variables JavaScript.

Operador	Ejemplo	Igual que
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

if...else
Ejecuta una sentencia si una condición especificada es evaluada como verdadera. Si la condición es evaluada como falsa, otra sentencia puede ser ejecutada.
Sintaxis

if (condición) sentencia1 [else sentencia2]

condición
Una expresión que puede ser evaluada como verdadera o falsa.
sentencia1
Sentencia que se ejecutará si condición es evaluada como verdadera. Puede ser cualquier sentencia, incluyendo otras sentencias if anidadas. Para ejecutar múltiples sentencias, use una sentencia block (**{ ... }**) para agruparlas.
sentencia2
Sentencia que se ejecutará si condición se evalúa como falsa, y exista una cláusula else. Puede ser cualquier sentencia, incluyendo sentencias block y otras sentencias if anidadas.

if...else

```
if (condición1)
   sentencia1
else if (condición2)
   sentencia2
else if (condición3)
   sentencia3
...
else
   sentenciaN
```

El bucle **For**
La sentencia **for** crea un bucle con 3 expresiones opcionales:

En este ejemplo se puede leer:

La expresión 1 establece una variable antes de que comience el bucle (que i = 0).
La expresión 2 define la condición para que se ejecute el bucle (i debe ser menor que 5).
La expresión 3 incrementa un valor (i++) cada vez que se ejecuta el bloque de código del bucle.

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i
+ "<br>";
}
```

El método getElementById
La forma más común de acceder a un elemento HTML es utilizar el id del elemento.

La propiedad innerHTML
La forma más sencilla de obtener el contenido de un elemento es utilizando la propiedad innerHTML.
La propiedad innerHTML es útil para obtener o reemplazar el contenido de elementos HTML.

La propiedad innerHTML puede usarse para obtener o cambiar cualquier elemento HTML, incluyendo <html> y <body>.

El objeto document HTML DOM

El objeto **document** representa tu página web.

Si quieres acceder a cualquier elemento de una página HTML, siempre empiezas accediendo al objeto **document**.

Metodo	Descripción
document.getElementById(*id*)	Buscar un elemento por su id
document.getElementsByTagName(*name*)	Buscar un elemento por su etiqueta name
document.getElementsByClassName(*name*)	Buscar un elemento por su clase

Obteniendo una referencia al objeto Element que representa un elemento.

```
<script>

   function iniciar() {
     var elemento = document.getElementById("subtitulo");
   }

 </script>
```

Leyendo los atributos de los elementos desde JavaScript.

```
<script>
   function iniciar() {
     var elemento = document.getElementById("subtitulo");
     alert("El id es: " + elemento.id);
   }
  </script>
</head>
<body onload="iniciar()">
  <section>
    <h1>Website</h1>
    <p id="subtitulo">El mejor sitio web!</p>
  </section>
</body>
</html>
```

Accediendo a un elemento por medio de su etiqueta.

```
<script>
    function iniciar() {
      var lista = document.getElementsByTagName("p");
      var elemento = lista[0];
      alert("El id es: " + elemento.id);
    }
  </script>
</head>
<body onload="iniciar()">
  <section>
    <h1>Sitio Web</h1>
    <p id="subtitulo">El mejor sitio web!</p>
  </section>
```

Buscando un elemento con el método querySelector().

```
<script>
   function iniciar() {
     var elemento = document.querySelector("section > p");
     alert("El id es: " + elemento.id);
   }
  </script>
</head>
<body onload="iniciar()">
  <section>
    <h1>Sitio Web</h1>
    <p id="subtitulo">El mejor sitio web!</p>
  </section>
```

Buscando un elemento dentro de otro elemento.

```
<script>
    function iniciar() {
      var elemprincipal = document.getElementById("seccionprincipal");
      var lista = elemprincipal.getElementsByTagName("p");
      var elemento = lista[0];
      alert("El id es: " + elemento.id);
    }
  </script>
</head>
<body onload="iniciar()">
  <section id="seccionprincipal">
    <h1>Sitio Web</h1>
    <p id="subtitulo">El mejor sitio web!</p>
  </section>
```

Asignando nuevos estilos al documento.

```
<script>
    function iniciar() {
      var elemento = document.getElementById("subtitulo");
      elemento.style.width = "300px";
      elemento.style.border = "1px solid #FF0000";
      elemento.style.padding = "20px";
    }
  </script>
</head>
<body onload="iniciar()">
  <section>
    <h1>Sitio Web</h1>
    <p id="subtitulo">El mejor sitio web!</p>
  </section>
```

Reemplazando la clase del elemento

```
<style>
   .supercolor {background: #0099EE;    }
   .colornegro {background: #000000;    }
  </style>
  <script>
   function cambiarcolor() {
     var elemento = document.getElementById("subtitulo");
     elemento.className = "colornegro";    }
  </script>
</head>
<body>
  <section>
   <h1>Sitio Web</h1>
   <p id="subtitulo" class="supercolor" onclick="cambiarcolor()">El mejor sitio web!</p>
  </section>
```

Activando y desactivando clases.

```
<style>
    .supercolor {
      background: #000000;
      color:white;
    }
  </style>
  <script>
    function cambiarcolor() {
      var elemento = document.getElementById("subtitulo");
      if (elemento.classList.contains("supercolor")) {
        elemento.classList.remove("supercolor");
      } else {
        elemento.classList.add("supercolor");
      }
    }
  </script>
```

Programación web en el entorno cliente

GRACIAS

MANUEL MACÍAS

TUTORIAS@MANUELMACIAS.ES

PROGRAMACIÓN JAVASCRIPT