UNIDAD 1.
Sintaxis básica del lenguaje

Desarrollo Web en Entorno Cliente

2º DAW

Contenidos

Integración de código JavaScript en HTML

Sentencia:

```
<script> </script>
```

Características de la etiqueta:

- Puede ir en <head> y/o en <body>
- Se puede utilizar las veces que se quiera

Dos modos de utilizarla:

- Código JavaScript entre la etiqueta de apertura y la de cierre.

```
<script>
     // Código
</script>
```

- Utilizar el atributo scr para incluir un archivo JavaScript

```
<script src="...."> </script>
```

No utilizar nunca las dos formas a la vez: el código que está entre las etiquetas, no se ejecuta.

```
<script src="...">
     // Código
</script>
```

También podemos insertar JavaScript en las propias etiquetas HTML:

```
<span onclick = "alert('¡¡Hola Usuario!!')">Haz click aquí</span>
```

Instrucciones de salida

Inicialmente se van a ver tres modos de “escribir” un texto en la pantalla o en una ventana de tipo pop-up. Aunque se volverán a estudiar más adelante, es necesario que desde el principio conozcamos estas instrucciones para hacer los primeros programas.

alert("cadena")

Muestra una ventana de tipo pop-up con el mensaje indicado como parámetro.
En realidad, es un método del objeto window (al ser el objeto por defecto, no es necesario especificarlo).
Las dos expresiones siguientes son equivalentes.

```
window.alert("cadena");
alert("cadena");
```

document.write("cadena")

Solo recomendable para pruebas. Ejemplo:

```
document.write("<h2>Titulo 2</h2>");
```

document.getElementById("id").innerHTML = "cadena"

Selecciona una elemento de HTML utilizando su ID y se asigna como contenido la "*cadena*". (Sobrescribe el contenido de la etiqueta, si lo tenía).
Solo se puede utilizar para elementos HTML que contengan texto (ej. no sirve para imágenes).
Ejemplo:

```
document.getElementById("miParrafo").innerHTML = "Parrafo nuevo";
```

console.log("cadena")

Muestra la "cadena" en la consola.
Ejemplo:

```
console.log("después del párrafo");
```

En los ejemplos anteriores, se ha pasado una "cadena". Esta cadena puede ser:

- Un string
- Un número (se convierte a cadena)
- Una operación matemática (se realiza y el resultado se convierte a cadena)

Ejemplo: Escribir el resultado de 3 + 2 (5) en la consola.

```
console.log(3 + 2);
```

Sintaxis de JavaScript

- **Valores**
  - **Literales**
    - Números enteros
    - Número decimal
    - Cadenas (entre comillas simples o comillas dobles)
  - **Variables**
    - Declaración:
      - var var;
      - var var1, var2, ...;
      - var var1 = expresion1;
      - var var1, var2 = expresion …;
- **Operadores**
  - Operador de asignación (=)
  - Operadores aritméticos, lógicos, etc.
- **Expresiones**
  - Pueden estar formadas por literales y variables, unidos con los operadores.
- **Palabras reservadas** (keywords)
- **Comentarios**
- **Identificadores:** nombre que se da a las variables y a las funciones
  - Pueden contener letras, números, y/o los caracteres $ y _
  - No pueden comenzar con número
  - No pueden ser las palabras reservadas
  - Son case-sensitive (las mayúsculas son diferentes de las minúsculas)
  - Utiliza el conjunto de caracteres **Unicode.**

- Recomendaciones:
  - Dos palabras: primeraSegunda
  - Recomendable utilizar identificadores descriptivos

Sentencias de JavaScript

Punto y coma al final de cada sentencia

Las sentencias deberían acabar con punto y coma (no es obligatorio, pero sí recomendable).
Se pueden escribir varias sentencias en la misma línea, separándolas por punto y coma.
Ejemplo:

```
var a= 1; var b= 2; var c=a+b;
```

Espacios en blanco

Espacios en blanco: no se tienen en cuenta, cuando hay más de uno.
Se recomienda utilizar espacios antes y después de los operadores =, +, -, *, /

Longitud de la línea

Las líneas pueden tener la longitud que queramos. Algunos programadores recomiendan no pasarse de 80 caracteres.

Sentencias que ocupan más de una línea

Si una sentencia no cabe en una línea, se puede continuar en la siguiente. Se recomienda dividir la línea después de un operador (ejemplo, operador de asignación)

Bloques de código

Son grupos de instrucciones que van entre dos llaves.
Se utilizan en funciones, bucles, condiciones, etc…

Palabras reservadas

Son: break, continue, switch, for, do … while, if … else, debugger, try … catch, var, let, const, function, return, etc.

Comentarios en JavaScript

Un comentario es texto que aparece en un programa, y que no se considera como código del programa; se entiende que es un comentario útil para el programador.

Comentarios de una línea

Todo lo que aparece en una línea después de `//`, se considera comentario.

Comentarios de bloque

Todo el texto que aparece entre **/*** y ***/**, se considera comentario.

Las variables en JavaScript

Nombre de las variables

*Ver "***Sintaxis de JavaScript***", apartado "Identificadores".*

Qué puede contener una variable

- Valores de tipo simple:
  - cadena (se entrecomillan con comillas simples o dobles)
  - números (enteros o decimales)
  - booleanos
- Objetos

Cómo declarar variables

Con la palabra reservada **var** se pueden declarar una o más variables separando la declaración de cada variable por comas.
Para cada variable, opcionalmente se le puede asignar un valor.

```
var identificador1 [ = valor1] {, identificador2 [ = valor2]}*
```

Ejemplo las siguientes declaraciones son correctas:

```
var nombre;
var nombre, apellido;
var nombre = "Ada";
var nombre = "Ada2", apellido;
var nombre = "Ada2", apellido = "Lovelace";
var nombre, apellido = "Lovelace";
var nombre = "Ada",
     apellido = "Lovelace";
```

Valor de las variables declaradas sin asignar un valor: undefined
Cuando se declara una variable, si no se le asigna un valor, tiene un valor indefinido (*undefined*).
Ejemplo:

```
var nombre;
```

Sentencia de asignación
Para asignar un valor a una variable se utiliza la sentencia:

```
nombreVariable = valor;
```

Qué valor se puede asignar a una variable
A una variable se le puede asignar (al crearla, o mediante una sentencia de asignación), un valor que puede ser una cadena, un número, o una expresión.

Ejemplo:

```
var nombreApellido = "Ada" + " " + "Lovelace";
var nacimiento = 1815;
var fallecimiento = nacimiento + 37;
```

En las expresiones con cadenas se pueden mezclar cadenas literales, variables de texto y variables de tipo cadena.
Ejemplo:

```
parrafo = nombreApellido + " nació en " + pais + ", el " +
nacimiento + " y murió el " + fallecimiento;
```

Declarar variables con la palabra reservada *let*★

Es lo recomendado desde la versión ES6. Resuelve algunos problemas que provocaba la declaración con **var**:
Ejemplo:

```
var persona = "Ada";
var persona = "Lovecode";
alert (persona);      //Resultado: Lovecode
let persona2 = "Charles";
let persona2 = "Babbage";  // Devolverá un error por haber
     definido la variable 2 veces
```

Modo estricto o *use strict*★

Para usar el código en "modo estricto", es decir, que no nos permita utilizar variables no declaradas. Incluir la sentencia **"use strict";**
Ejemplo1:

```
let persona = "Ada";
nacimiento = "1815";
alert (persona + " nació en " + nacimiento);     // Devolverá:
     Ada nació en 1815
```

Ejemplo2:

```
"use strict";
let persona = "Ada";
nacimiento = "1815";
alert (persona + " nació en " + nacimiento);     // Devolverá
     error por haber definido la variable 2 veces
```

Las constantes en JavaScript★

Se declaran con la palabra reservada **const**:
Ejemplo:

```
const SALUDO = "Hola";
```

★ *EcmaScript 2015 o ES6 o JS ES6 o JavaScript 6.*

Operadores aritméticos

Sus operandos son de tipo numérico y devuelven un valor numérico.

Operadores binarios

+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo o resto
**	Exponenciación; asociativo por la derecha (*EcmaScript 2015 o ES6 o JS ES6 o JavaScript 6)*

Operadores unarios

++	Suma 1 Se utiliza delante o detrás de una variable
--	Resta 1 Se utiliza delante o detrás de una variable
+	Más unario Convierte su operando en número
-	Menos unario Devuelve su operando negado Convierte el operando en número

Operadores de asignación

Permiten asignar el valor de una expresión a una variable.

=	Asignar
+=	Sumar lo que hay a la derecha
-=	Restar lo que hay a la derecha
*=	Multiplicar por lo que hay a la derecha
/=	Dividir por lo que hay a la derecha
%=	Obtener el resto (módulo) de dividir entre lo que hay a la derecha

Operadores de cadena

Operadores de cadena

+	Concatena
+=	Concatenar lo que hay a la derecha

Suma de cadenas y números

En realidad las cadenas y los números no se pueden sumar porque son de diferentes tipos. Cuando indicamos una suma con números y cadenas, JavaScript entiende que queremos sumar cadenas y convierte internamente los números a cadenas.
Ejemplo

```
var resultado = 4 + "Ada";       // resultado contendrá "4Ada"
```

Sin embargo, si en la suma hay operadores de tipo numérico antes de la cadena, JavaScript primero realizará la suma de los de tipo numérico y después lo convertirá a cadena (por la precedencia de operadores (entre operadores iguales, se ejecutan primero los que aparecen antes)

```
var resultado = 4 + 5 + "Ada";   // resultado contendrá "9Ada"
var resultado = "Ada" + 4 + 5;   // resultado contendrá "Ada45"
var resultado = "Ada" + (4 + 5); // resultado contendrá "Ada9"
```

Operadores de comparación

Se utilizan para determinar la igualdad o desigualdad entre dos expresiones.
Devuelve un valor booleano que puede ser *true* o *false*.

==	Igualdad
===	Igualdad estricta true: si ambos operadores tienen el mismo valor y son del mismo tipo
!=	Desigualdad
!==	Desigualdad estricta true: si ambos operadores tienen distinto valor o tienen distinto tipo
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

Operadores lógicos

Se utilizan para determinar la lógica de variables o de operaciones.
Se utilizan con operandos de tipo lógico o boolean.
El resultado es verdadero o falso.

&&	and lógico
||	or lógico
!	not lógico

Operador ternario

? :	condicion? valor1 : valor2

Se evalúa la condición. Si es correcta, esta expresión devuelve el valor “resultado_verdadero”; en otro caso devuelve el valor “resultado_falso”.
Ejemplo

```
var edad = 12;
var puedeVotar = (edad >=18 ? "Puede votar": "No puede votar");
```

La expresión anterior sería equivalente a:

```
if (edad >= 18)
     puedeVotar = "Puede votar";
else
     puedeVotar = "No puede votar";
```

Operadores de tipos

Son: typeof y instanceof.

typeof

Devuelve un string que contiene el tipo de dato de una expresión.
El tipo de datos se devuelve como un string en minúsculas.

Sintaxis: Admite dos formas.

```
typeof expresión --> string
typeof (expresión) --> string
```

Puede devolver:

- string
- number
- boolean

- object (para array, registro, date, objetos, variable definida como null)
- function
- undefined (para variables no definidas)

Ejemplos:

```
var cadena="Ada";
tipo = typeof cadena;            // Devuelve "string"
tipo2 = typeof "Ada Lovelace";   // Devuelve "string"
```

instanceof

Devuelve un valor booleano que indica si un objeto es una instancia de otro especificado.

Sintaxis:

```
objeto1 instanceof objeto2   --> boolean
```

Ejemplos:

```
var animales = ["perro", "gato", "hamster"];
alert(animales instanceof Object);    //devuelve true
alert(animales instanceof Array);     //devuelve true
alert(animales instanceof Number);    //devuelve false
alert(animales instanceof String);    //devuelve false
```

Precedencia de los operadores

Los operadores tienen la siguiente precedencia (de mayor a menor):

Agrupar (paréntesis)	( )	n/a
Incremento	++, --	n/a
Operadores unarios	+ unario, - unario, !	derecha a izquierda
Multiplicación, división, resto	*, /, %	izquierda a derecha
Suma y resta	+, -	izquierda a derecha
Relacionales	<, <=, >, >=	izquierda a derecha
Igualdad	==, !=, ===, !==	izquierda a derecha
AND lógico	&&	izquierda a derecha
OR lógico	||	izquierda a derecha
Condicional	.. ? … : ...	derecha a izquierda
Asignación	=, +=, -=, *=, /=, %=...	derecha a izquierda
Coma	,	izquierda a derecha

Tipos de datos

Tipos de datos primitivos

- boolean
- number
- string
- null
- undefined

Tipos de datos no primitivos

- object

boolean

Solo pueden tener uno de dos valores posibles: *true* / *false*.

number

El tipo Number incluye todos los números: enteros, reales, etc.
Tipos de números:

- **Enteros**: Ej. 234;
- **Decimales**: Ej. 234.25;
- **Decimales en notación científica**. Esta notación se utiliza para representar números muy grandes o muy pequeños Ej. 12e5, 1.1234e-3;

Algunos valores especiales que pueden tener las variables de tipo number:

- **NaN** (Not a Number)
  Ejemplo: al intentar sumar un “undefined” y un number

```
var a;
a + 2;      // devuelve NaN
```

- **Infinity** y **–Infinity**: Al dividir un número distinto de 0 entre 0.

string

Se definen entre comillas simples o dobles.
Ejemplo:

```
"Ada"
'Ada'
```

Para que una cadena contenga comillas simples, habrá que rodearla con comillas dobles, y viceversa.
Ejemplo:

```
"Ada es la 'mejor'"
'Ada es la "mejor"'
```

Como hemos visto, cuando se suma (operador +) una cadena con un dato de otro tipo, ese dato se convierte en cadena, y después se concatenan.
Así, las expresiones siguientes son equivalentes:

```
"Ada nació en " + 1815
"Ada nació en " + "1815"
"Ada nació en 1815"
```

Para indicar que una variable es de tipo string, pero está vacía, se le asigna una cadena vacía.
Ejemplo:

```
var cadena4 = "";
```

Valor null

Solo puede contener el valor null.
Ejemplo:

```
var animal = null;
```

null se evalúa

- como false en operaciones lógicas,
- y como 0 en operaciones aritméticas

Algunas expresiones:

```
typeof null;;    // Devuelve "object"
```

Valor undefined

Una variable a la que no se ha asignado un valor, tiene el valor undefined.
Se puede utilizar para quitar la definición de un objeto (poco habitual, pero se puede hacer):

```
var cadena = undefined;
```

undefined se evalúa

- como false en operaciones lógicas.

Algunas expresiones:

```
typeof undefined;     // Devuelve undefined
null === undefined;   // Devuelve false
null == undefined;    // Devuelve true
```

Arrays

Los arrays son un tipo de objetos.
Un array pueden contener datos de diferentes tipos (ej. números, cadenas, etc.).

Declarar una variable de tipo array
Sintaxis:

```
var variable = [ { valor {, valor }*} ];
```

Ejemplo:

```
var lista = [];
lista = [1, 2, 5, 7];
var lista2 = [1, "Ada", 3, 4, "Lovelace"];
```

Acceder a un elemento del array
Como en otros lenguajes, el índice empieza en 0.
Sintaxis:

```
variable [índice];
```

Ejemplo:

```
lista[0];
lista[3];
```

Objetos

En JavaScript se pueden crear objetos propios (se verá en profundidad más adelante).

Declarar una variable de tipo objeto
Sintaxis:

```
var variable = {clave: valor {, clave: valor}*};
```

Ejemplo:

```
var animal = {
     nombre: "Lola",
     tipo: "Hamster",
     raza: "Ruso",
     edad: 1
};
```

Acceder a las propiedades de un objeto
Sintaxis:

```
objeto.propiedad;
```

Ejemplo:

```
console.log(animal.raza);
```

Plantillas de cadena de texto (*Template Literals*)★

Las plantillas de texto son literales de texto que permiten generar una cadena a partir de una plantilla. La plantilla puede incluir expresiones (ej. nombres de variables o expresiones aritméticas), y puede incluir también saltos de línea.
Una plantilla de texto se escribe entre comillas o tildes invertidas: `.

Obtener cadenas de más de una línea
Los caracteres de fin de línea encontrados son parte de la plantilla de cadena de texto. En el caso de cadenas de texto normales, esta es la sintaxis necesaria para obtener cadenas de más de una línea:

```
//ES5
console.log("línea 1 de cadena de texto\n\
línea 2 de cadena de texto");

//ES6: para obtener el mismo resultado utilizando plantillas:
console.log(`línea 1 de la cadena de texto
```

★ *EcmaScript 2015 o ES6 o JS ES6 o JavaScript 6.*

```
línea 2 de la cadena de texto`);
```

Expresiones dentro de una plantilla

Dentro de una cadena delimitada por comillas invertidas (``) se pueden incluir expresiones entre llaves y precedidas por un símbolo dólar (${}). Las expresiones se evalúan y el resultado forma parte de la cadena.

Ejemplo: Mostrar por consola:

"Quince es 15 y
no 20."

```
// ES5
var a = 5;
var b = 10;
console.log("Quince es " + (a + b) + " y\nno " + (2 * a + b) +
".");

// ES6
var a = 5;
var b = 10;
console.log(`Quince es ${a + b} y\nno ${2 * a + b}.`);
```

Funciones. Introducción

Es un bloque de código que se utiliza para ejecutar una tarea en particular.

Nombre de las funciones

Ver "Sintaxis de JavaScript", apartado "Identificadores".

Definir una función

Sintaxis

```
function nombreFuncion (parámetro1, parámetro2, … , parámetroN) {
      código
}
```

Ejemplos:

```
function saludo() {
      alert("Hola");
}
function producto(a, b) {
      return a * b;
}
```

Las funciones pueden devolver datos o no.
Para que devuelva algo, utilizamos la instrucción **return**. (Si no utilizamos esa instrucción, no devuelve nada).

Llamar a una función

Sintaxis

```
nombreFuncion (parámetros);
```

Si no utilizamos los paréntesis, no se llama a la función.

Ejemplo:

```
saludo();
var res = producto(3, 4);
alert(res);
alert(producto( 3, 4 ));
```

Funciones. Parámetros y argumentos

Una función puede tener cero o más parámetros.
Parámetro: Nombres que aparecen en la definición de una función.
Argumento: valor que se pasa (y que recibe) una función.
Reglas de los parámetros

- Como hemos visto, no se especifica el tipo de los parámetros.
- No se verifican los tipos de los argumentos.
- No se comprueba el número de los argumentos recibidos.

Parámetros por defecto[⋆]

¿Qué ocurre si llamamos a una función con menos argumentos de los definidos?
Los parámetros que no tienen valor, toman el valor "undefined". La versión ES6 introduce la opción de poder inicializar los parámetros formales de una función con un valor por defecto.

Ejemplo:

```
// ES5
function producto(a, b) {
      if (typeof b === undefined) b = 1;
      return a * b;
}
var resultado = suma(4);
// ES6
function producto(a, b = 1) {
      return a * b;
}
var resultado = suma(5);
```

Parámetros por exceso★

¿Qué ocurre si llamamos a una función con más argumentos de los definidos?
Los valores que nos llegan se pueden capturar a través de un objeto incluido en la función llamado **arguments** (es un objeto iterable, formado por pares objeto / valor). La versión ES6 introduce los **parámetros REST** que nos permite representar un número indefinido de argumentos con un array. Para ello utilizamos el operador **...** delante del nombre del array y siempre en el último lugar de la lista de parámetros.

Ejemplo: función que muestre los parámetros que se han pasado.

```
//ES5
function valores(a, b) {
    console.log("ARGUMENTS");
    console.log("a=" + a);
    console.log("b=" + b);
    console.log("El número de argumentos es " + arguments.length);
    console.log("Primer elemento de la lista: " + arguments[0]);
    console.log("Todos los elementos de la lista: ", arguments);
}
valores(4, 6, 8, 2, 7, 5);
//ES6
function valores2(a, b, ...elementos) {
    console.log("REST");
    console.log("a=" + a);
    console.log("b=" + b);
    console.log("El número de argumentos es " + elementos.length);
    console.log("Primer elemento de la lista: " + elementos[0]);
    console.log("Todos los elementos de la lista: ", elementos);
}
valores2(4, 6, 8, 2, 7, 5);
```

⋆ *EcmaScript 2015 o ES6 o JS ES6 o JavaScript 6.*

Funciones anónimas

Son funciones que no tienen nombre.

Declarar una función anónima

Se declaran como las otras funciones, pero sin nombre.
Las funciones anónimas se pueden asignar a una variable.
Ejemplo:

```
var producto = function (a, b) {
      return a * b;
};
alert(producto(5, 4);
```

equivale a:

```
function producto (a, b) {
      return a * b;
}
alert(producto(5, 4);
```

Llamar a una función anónima

Sintaxis:

```
nombreVariable(parámetros);
```

Se podría asignar una función a todas las variables que queramos.
Ejemplo:

```
producto(3, 4);
```

Funciones anónimas autoinvocadas

Se utilizan mucho cuando trabajamos con eventos.
Para que una función se ejecute automáticamente, sin ser llamada, al final de la función, es decir, después de la última llave, tenemos que incluir los paréntesis abrir y cerrar.
Y para que se ejecute como una expresión, debemos meter toda la sentencia entre dos paréntesis.
Sintaxis:

```
(function(<parámetros>) {<instrucciones>}());
```

Ejemplo:

```
(function() {alert("Hola");}());
```

Funciones flecha (*arrow functions*)*

Son una alternativa compacta a una función convencional.
No son adecuadas para ser utilizadas como métodos y no pueden ser usadas como constructores. Son siempre anónimas.

* *EcmaScript 2015 o ES6 o JS ES6 o JavaScript 6.*

Sintaxis básica

En lugar de la palabra clave **function**, utiliza una flecha (=>) formada por el signo igual (=) seguido del signo mayor que (>).
La flecha va detrás de la lista de parámetros, y seguida del cuerpo de la función:

```
(param1, param2, …, paramN) => {sentencias}
```

Ejemplo:

```
var producto = (a, b) => {
     return a * b;
};
alert(producto(5, 4));
```

equivale a:

```
var producto = function (a, b) {
     return a * b;
};
alert(producto(5, 4));
```

Si la función tiene una sola sentencia que devuelve un valor, podemos eliminar los corchetes y la palabra clave **return**:

```
(param1, param2, …, paramN) => {return expresión;}
(param1, param2, …, paramN) => expresión;
```

Ejemplo:

```
var producto = (a, b) => a * b;
alert(producto(5, 4));
```

Los paréntesis son opcionales si tenemos un solo parámetro:

```
(únicoParam) => {sentencias}
únicoParam => {sentencias}
```

Ejemplo:

```
var hola = (persona) => "Hello " + persona;
alert(hola("Ada"));
```

equivale a:

```
var hola = persona => "Hello " + persona;
alert(hola("Ada"));
```

Los paréntesis son obligatorios si una función no tiene parámetros:

```
() => {sentencias}
```

Ejemplo:

```
var hola = () => "Hola";
alert(hola());
```

Ámbito de las variables

Zona del programa en el que la variable puede trabajar.

Variables locales

Características:

- Se definen con **var** o **let** dentro de una función.
- Tienen ámbito local en la función, es decir solo se pueden utilizar dentro de ella, por tanto:
  - Podemos declarar variables con el mismo nombre en diferentes funciones.
  - Desaparecen cuando finaliza la función.

Ejemplo:

```
function saludar () {
      var saludo = "Hola";
      alert(saludo);
}
saludar();
```

Variables globales

Características:

- Se definen con **var** o **let** fuera de las funciones.
- Tienen ámbito global (en toda la página), por tanto:
  - Son accesibles desde fuera y dentro de las funciones.
  - Desaparecen cuando se sale de la página.
  - Son propiedades del objeto window, que es el objeto global de las páginas web.

Ejemplo:

```
var despedida = "Adiós";
function despedir () {
      alert(despedida);
}
despedir();
```

Variables automáticamente globales

Se crean cuando asignamos un valor a una variable que no ha sido declarada antes.
Ejemplo:

```
function preguntar() {
      pregunta="De qué color es el caballo blanco de Santiago? ";
}
alert(pregunta);
```

Variables de bloque*

Las variables declaradas con **var:**

* *EcmaScript 2015 o ES6 o JS ES6 o JavaScript 6.*

- **No** pueden tener ámbito de bloque
- **Sí** se podrá acceder a las variables declaradas dentro de un bloque {sentencias} desde fuera del mismo
- **Sí** nos permitirá declarar una variable que se ha utilizado anteriormente**.**

Ejemplo:

```
function ambito() {
      var miVariable = "dentro";
      alert(miVariable);
}
alert(miVariable);    //Sí que puedo acceder a la variable
sinDeclarar = "fuera";
var sinDeclarar; //Sí que puedo declararla después de usada
```

Las variables declaradas con la palabra **let:**

- **Sí** que pueden tener ámbito de bloque.
- **No** se podrá acceder a las variables declaradas dentro de un bloque {sentencias} desde fuera del mismo
- **No** nos permitirá declarar una variable que se ha utilizado anteriormente**.**

Ejemplo:

```
function ambito() {
      let miVariable = "dentro";
      alert(miVariable);
}
alert(miVariable);    //No puedo acceder a la variable
sinDeclarar = "fuera";
let sinDeclarar; //No que puedo declararla después de usada
```

Variables locales y globales con el mismo nombre

Si se utiliza en la función, se utiliza la variable local a la función.
Si se utiliza fuera de la función, se utiliza la variable global.
Ejemplo:

```
var miVariable = "fuera";
function ambito() {
      var miVariable = "dentro";
      alert(miVariable);
}
alert(miVariable);    //Devuelve fuera
ambito();  //Devuelve dentro
alert(miVariable);    //Devuelve fuera
```

Condicionales. Sentencias if-else

En esta sentencia:

- <condicion> es una expresión que se evalúa como true o false.
- Los bloques else if pueden aparecer o no.
- El último bloque else se puede utilizar o no.
- En los bloques de código que están entre llaves, se pueden suprimir las llaves si solo hay una instrucción. Sin embargo, se recomienda utilizarlas por claridad del código.

Sintaxis: if else if … else

```
if (<condición>) {
     // Instrucciones si la condición es verdadera
} [else if (<condicion2>) {
     // Instrucciones si la condición2 es verdadera
}][else {
     // Instrucciones si las condiciones anteriores son falsas
}]
```

Ejemplo 1:

```
var nota = 3;
if (nota >= 5) {
     alert ("Has aprobado");
}
```

Ejemplo 2:

```
var nota = 3;
if (nota >= 5) {
     alert ("Has aprobado");
} else {
     alert ("Has suspendido");
}
```

Ejemplo 3:

```
var nota = 3;
if ( nota < 5 ) {
     alert ("Has suspendido");
} else if ( nota == 5 ) {
     alert ("Has sacado un suficiente");
} else if ( nota == 6 ) {
     alert ("Has sacado un bien");
} else if ( nota == 7 || nota == 8) {
     alert ("Has sacado un notable");
} else  {
     alert ("Has sacado un sobresaliente");
}
```

Condicionales. Sentencia switch

Se utiliza para realizar diferentes acciones, dependiendo del valor de una expresión.

Sintaxis:

```
switch (<expresión>) {
      case <valor1>:
            // Instrucciones si <expresión> se evalúa al <valor_1>
            [break;]
      case <valor_n>:
            // Instrucciones si <expresión> se evalúa al <valor_n>
            [break;]
      default:
            // Instrucciones para el resto de los casos
            [break;]
}
```

Descripción:

- Primero evalúa la <**expresión**> que está después de la palabra reservada **switch** y se obtiene su valor (ej. la expresión 2 + 3 se evalúa y se obtiene el valor 5).
- Se busca la primera cláusula **case** cuyo valor coincida con el valor de la expresión (**utilizando comparación estricta**):
    - Se transfiere en control a la sentencia que esté a continuación de la cláusula **case**.
    - Es necesario utilizar una sentencia **break** para salir del **switch**; si no se hace, se ejecutan todas las instrucciones que estén a continuación, aunque correspondan a otro case.
    - La etiqueta **default** se utiliza para encabezar el código que se ejecuta si no se ha encontrado el valor en ninguna sentencia case. Por convención, la cláusula **default** es la última cláusula, pero podría colocarse en otro lugar (si tiene sentido).

Ejemplo:

```
var nota = 3;
var mensaje = "";
switch (nota) {
      case 0:
      case 1:
      case 2:
      case 3:
      case 4:
            mensaje = "suspenso";
            break;
      case 5:
            mensaje = "suficiente";
            break;
      case 6:
            mensaje = "bien";
            break;
      case 7:
      case 8:
            mensaje = "notable";
            break;
```

```
      case 9:
      case 10:
            mensaje = "sobresaliente";
            break;
      default:
            mensaje = "El valor no es válido";
}
alert(mensaje);
```

Repeticiones. Bucle for

Sintaxis:

```
for (<sentencia1>; <sentencia2>; <sentencia3>) {
      // Código que se repite
}
```

Donde:

- <sentencia1>
  - se ejecuta una vez, antes de que empiece el bloque de código
  - puede contener varias sentencias separadas por comas
  - esta sentencia se puede omitir, pero debemos haber inicializado la/s variable/s
- <sentencia2>
  - es la condición que se evalúa en cada iteración; si el resultado es verdadero (true), se ejecuta el código
  - si se omite, el bloque de código se ejecuta siempre; para salir del bucle habrá que utilizar la sentencia break
- <sentencia3>
  - se ejecuta después de la ejecución del código
  - puede contener varias sentencias separadas por comas
  - se puede omitir, pero debemos incrementar o decrementar el valor de la variable dentro del bucle

Ejemplo:

```
var i;
for (i=0; i <= 5; i++) {
      alert(i);
}
```

El siguiente código es equivalente al anterior:

```
var i = 0;
for (; i <= 5;) {
      alert(i);
      i++;
}
```

En este ejemplo, se realizan varias operaciones al principio del bucle, y después de cada iteración:

```
var m, n;
for (n=0, m=10; n <=10; n++, m--) {
      alert("n vale " + n + "; m vale " + m);
}
```

Repeticiones. Bucle for in

Se utiliza para recoger las propiedades y métodos de un objeto.
Después, se puede acceder al contenido de la propiedad o método, utilizando la notación de corchetes:

```
objeto[propiedad]
```

Sintaxis:

```
for (<nom_propiedad > in <objeto>) {
      // sentencias
}
```

Ejemplo:

```
var animal= {
      nombre: "Lola",
      tipo: "Hamster",
      raza: "Ruso",
      edad:1}
var prop;
for (prop in animal) {
      alert(prop);          // Mostrará nombre, tipo, raza, edad
      alert(animal[prop]);  // Mostrará Lola, Hamster, Ruso, 1
}
```

Repeticiones. Bucle while

Sintaxis:

```
while (<condicion>) {
      //Código
}
```

Ejemplo:

```
var i = 0;
while (i < 10) {
      alert (i++);
}
```

Repeticiones. Bucle do while

Sintaxis:

```
do {
      //Código
} while (<condición>);
```

Ejemplo:

```
var i = 0;
do {
      alert (i++);
} while (i < 10);
```

Saltos. Sentencias break y continue

Se utilizan dentro de bloques de código. Permiten cambiar el orden normal de ejecución del código.

break

Sale del bloque de código: la siguiente sentencia a ejecutar será la que haya después de la llave de cerrar.
Lo hemos utilizado en la sentencia **switch**, y también en la sentencia **for**. Se puede utilizar en cualquier bucle.

continue

Salta a la siguiente iteración del bucle.
Por ejemplo, si está dentro de un bucle **for**, se ejecutará la sentencia3 (incremento) y la sentencia2 (condición).
Si está dentro de un **while** o **do while**, irá a la condición.

etiquetas

Preceder una sentencia con el nombre de la etiqueta seguido de dos puntos.
Sintaxis:

```
etiqueta: {
      //Código
}
```

Si el código está formado por una sola línea, se puede escribir sin llaves.

Ejemplo:

```
var animales = ["perro", "gato", "hamster"];
verAnimales: {
      alert(animales[0]);
      alert(animales[1]);
      alert(animales[2]);
}
```

Uso de break con etiquetas
Se puede utilizar el break para salir del bloque de código. Para ello, hay que utilizar el break con el nombre de la etiqueta
Sintaxis:

```
break etiqueta;
```

Ejemplo:

```
verAnimales: {
      alert(animales[0]);
      alert(animales[1]);
      break ver_animales;
      alert(animales[2]);
}
```