UNIDAD DIDACTICA 03

CARACTERISTICAS DEL LENGUAJE JAVASCRIPT.

Objetivos.

- ✓ Conocer las principales características del lenguaje JavaScript.
- ✓ Dominar la sintaxis básica del lenguaje.
- ✓ Comprender y utilizar los distintos tipos de variables y operadores presentes en el lenguaje JavaScript.
- ✓ Conocer las diferentes sentencias condicionales de JavaScript y saber realizar operaciones complejas con ellas.

1.- EL lenguaje JavaScript: sintaxis.

El lenguaje JavaScript tiene una sintaxis muy parecida a la de Java o a la de C++. La sintaxis especifica aspectos, como los caracteres que se deben utilizar para definir comentarios, la forma de los nombres de las variables o el modo de separar diferentes instrucciones del código.

Uso de mayúsculas y minúsculas:

Uno de las primeras dificultades que nos encontramos cuando se empieza a programar en JavaScript es que este lenguaje distingue entre mayúsculas y minúsculas. A diferencia del HTML. Esta regla es importante cuando utilizamos variables, objetos, funciones o cualquier otro símbolo del lenguaje.

Por ejemplo, no es lo mismo utilizar la función `alert()` que `Alert()`. La primera, nos muestra una ventana emergente del navegador, y la segunda no existe, a menos que no la defina el programador.

Comentarios en el código.

Tabulaciones y saltos de línea.

JavaScript ignora los espacios, las tabulaciones y los saltos de línea presentes entre los símbolos del código. En el momento en que los programas empiezan ser complejos, podemos apreciar la utilidad de emplear tabulaciones y los saltos de línea adecuados para mejorar la presentación y la legibilidad del código.

El punto y coma.

Normalmente suele insertarse el punto y coma (`;`) al final de cada instrucción de JavaScript. Este signo tiene la utilidad de separar y diferenciar cada instrucción, por lo que solo sería obligatorio si ponemos dos instrucciones en la misma línea.

Palabras reservadas en JavaScript

Las siguientes son palabras reservadas y no pueden ser utilizadas como variables, funciones, métodos o identificadores de objetos. Las siguientes son reservadas como palabras claves existentes por la especificación ECMAScript:

Break	Case	Class	Catch	Const	Continue	Debugger
Else	Export	Extends	Finally	For	Function	If
Let	New	Return	Super	Switch	This	Throw
Void	While	With	Vield	Enum	Implements	Package
Public	Abstract	Boolean	Byte	Char	Doublé	Final
Long	Native	Short	Syncronized	Transient	volatile	Null
Do	Var	Int	In	Interface	false	Import
Instanceof	Private	Delete	Typeof	Goto	Default	Try
Static	Float	True				

2.- Las variables.

La forma más conveniente de trabajar con datos en un script, es asignando primeramente los datos a una variable.

Una variable es un espacio de memoria donde se almacena un dato, espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestro programa.

El tiempo que dicha información permanecerá almacenada, dependerá de muchos factores. En el momento que el navegador limpia la ventana o marco, cualquier variable conocida será eliminada.

Dispones de dos maneras de crear variables en JavaScript: una forma es usar la palabra reservada **var** seguida del nombre de la variable. Por ejemplo, para declarar una variable edad, el código de JavaScript será:

```
var edad;
```

Otra forma consiste en crear la variable, y asignarle un valor directamente durante la creación:

```
var edad = 38;
```

O bien, podríamos hacer:

```
var edad;
edad = 38; // Ya que no estamos obligados a declarar la variable pero es una buena práctica el
hacerlo.
var altura, peso, edad: // Para declarar más de una variable en la misma línea.
```

La palabra reservada **`var`** se usa para la declaración o inicialización de la variable en el documento.

Una variable de JavaScript podrá almacenar diferentes tipos de valores, y una de las ventajas que tenemos con JavaScript es que no tendremos que decirle de qué tipo es una variable u otra.

A la hora de dar nombres a las variables, tendremos que poner nombres que realmente describan el contenido de la variable. No podremos usar palabras reservadas, ni símbolos de puntuación en el medio de la variable, ni la variable podrá contener espacios en blanco. Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). No podremos utilizar caracteres raros como el signo `+`, `un espacio`, `%` , `$`, etc. en los nombres de variables, y estos nombres no podrán comenzar con un número.

Si queremos nombrar variables con dos palabras, tendremos que separarlas con el símbolo "_" o bien diferenciando las palabras con una mayúscula, por ejemplo:

```
var mi_peso;
var miPeso;  // Esta opción es más recomendable, ya que es más cómoda de escribir.
```

Deberemos tener cuidado también en no utilizar nombres reservados como variables. Por ejemplo, no podremos llamar a nuestra variable con el nombre de `return` o `for`.

Un aspecto a tener en cuenta es el ámbito de las variables o lugar donde están disponibles. Por lo general cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, como JavaScript se define dentro de una página web, las variables que declaremos en la página estarán accesibles dentro de ella.

En JavaScript no podremos acceder a variables que hayan sido definidas en otra página. Por tanto, la propia página donde se define es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página. Las variables según su ámbito pueden ser:

- **Variables globales**: son las que están declaradas en el ámbito más amplio posible, que en JavaScript es una página web. Para declarar una variable global a la página simplemente lo haremos en un script, con la palabra `var`. Son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página, incluidos los manejadores de eventos.
- **Variables locales**: podremos declarar variables en lugares más acotados, como por ejemplo una función. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función. Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle.

No hay problema en declarar una variable local con el mismo nombre que una global, en este caso la variable global será visible desde toda la página, excepto en el ámbito donde está declarada la variable local ya que en este sitio ese nombre de variable está ocupado por la local y es ella quien tiene validez.

Ejemplo:

```
<SCRIPT>
    var variableGlobal
</SCRIPT>
<SCRIPT>
    function miFuncion (){
       var variableLocal
    }
</SCRIPT>
```

3.- Tipos de datos.

Los tipos de datos o valores son clases de forma que cada variable o constante que definamos pertenecerá a uno de ellos. En JavaScript no es necesario definir el tipo de una variable al declararla, sino que será el propio JavaScript el que determine a cual corresponde en función del valor asignado. Gracias a esto, una variable puede cambiar su tipo.

A continuación, se muestran los tipos de datos soportados en JavaScript:

Tipos de datos soportados por JavaScript		
Tipo	**Ejemplo**	**Descripción**
Cadena	“Hola mundo”	Una serie de caracteres dentro de unas comillas.
Número	9.45	Un número sin comillas dobles.
Booleano	True	Un valor verdadero o falso.
Null	Null	Sin contenido, simplemente es el valor Null
Object		Es un objeto software que se define por sus propiedades y métodos (los arrays son objetos)
Function		La definición de una función.

El contener solamente este tipo de datos, simplifica mucho las tareas de programación, especialmente aquellas que abarcan tipos incompatibles entre números.

3.1.- Tipo de datos numérico.

En este lenguaje solo existe un tipo de datos numérico, al contrario que ocurre con la mayoría de los lenguajes más conocidos. Todos los números son por tanto del tipo numérico, independientemente de la precisión que tenga o si son números reales o enteros. Los números enteros son números que no tiene coma, como 3 o

366. Los números reales son números fraccionarios, como 2,69 o 0,25, que también se pueden escribir en notación científica, por ejemplo 2,48e12

Con JavaScript también podemos escribir números en otras bases, como la hexadecimal. Las bases son sistemas de numeración que utilizan más o menos dígitos para escribir los números. Existen tres bases con las que podemos trabajar:

- Base 10, es el sistema que utilizamos habitualmente, el sistema decimal. Cualquier número, por defecto, se entiende que está escrito en base 10.
- Base 8, también llamado sistema octal, que utiliza dígitos del 0 al 7. Para escribir un número en octal basta con escribir ese número precedido de un 0, por ejemplo 045. Está obsoleto.
- Base 16 o sistema hexadecimal, es el sistema de numeración que utiliza 16 dígitos, los comprendidos entre el 0 y el 9 y las letras de la A a la F, para los dígitos que faltan. Para escribir un número en hexadecimal debemos escribirlo precedido de un cero y una equis (0x), por ejemplo, 0x3EF.

3.2.- Tipo booleano.

Los tipos de datos bolean, booleano, son un `true` o un `false` y se usan para tomar decisiones. Los únicos valores que podemos poner en las variables booleanas son `true` y `false`.

```
Si una variable es verdadero entonces
     Ejecuto unas instrucciones
Si no
     Ejecuto otras
```

Ejemplo:

```
<SCRIPT TYPE=”text/javascript”>
 //Declaración de datos de tipo lógico o booleano
        varesVerdadero = true;
        varesFalso = false;
        var COMPROBAR = true;
</SCRIPT>
```

CUIDADO!!!! *JavaScript distingue entre mayúsculas y minúsculas, por tanto, True o TRUE no lo son valores reconocidos por JavaScript.*

3.3.- Cadena de caracteres.

También denominadas ***string***. Este tipo consta de un texto con cualquier carácter (letras, números y símbolos). Las cadenas de caracteres se escriben en el código entre comillas dobles (`"”`) o simples (`''`), pero nunca mezcladas. Lo más habitual es utilizar las comillas dobles para representar un texto.

```
miTexto = "Pepe se va a pescar"
miTexto = '23%%$ Letras & *--*'
```

Todo lo que se coloca entre comillas, como en los ejemplos anteriores es tratado como una cadena de caracteres independientemente de lo que coloquemos en el interior de las comillas.

Caracteres escape en las cadenas de texto:

Hay una serie de caracteres especiales que sirven para expresar en una cadena de texto determinados controles como puede ser un salto de línea o un tabulador. Estos son los caracteres de escape y se escriben con una notación especial que comienza por una contra barra (una barra inclinada al revés de la normal) y luego se coloca el código del carácter a mostrar.

Un carácter muy común es el salto de línea, que se consigue escribiendo n. Otro carácter muy habitual es colocar unas comillas, pues si colocamos unas comillas sin su carácter especial nos cerrarían las comillas que colocamos para iniciar la cadena de caracteres. Las comillas las tenemos que introducir entonces con " o ' (comillas dobles o simples). Existen otros caracteres de escape, que veremos en la tabla de abajo más resumidos, aunque también hay que destacar como carácter habitual el que se utiliza para escribir una contrabarra, para no confundirla con el inicio de un carácter de escape, que es la doble contrabarra.

Tabla con todos los caracteres de escape:

Secuencia de escape	Descripción
\”	Comillas dobles
\’	Comillas simples
\\	Barra inversa o contrabarra
\n	Salto de línea
\t	Tabulación horizontal
\r	Retorno de carro
\b	Retroceso
\f	Salto de página
\v	Tabulación vertical

3.4.- Conversiones de tipos de datos.

Aunque los tipos de datos en JavaScript son muy sencillos, a veces nos podemos encontrar con casos en los que las operaciones no se realizan correctamente, y eso es debido a la conversión de tipos de datos. JavaScript intenta realizar la mejor conversión cuando realiza esas operaciones, pero a veces no es el tipo de conversión que nos interesa.

Por ejemplo, cuando intentamos sumar dos números:

```
4 + 5 // resultado = 9
```

Si uno de esos números está en formato de cadena de texto, JavaScript lo que hará es intentar

convertir el otro número a una cadena y los concatenará, por ejemplo:

```
4 + "5" // resultado = "45"
```

Otro ejemplo podría ser:

```
4 + 5 + "6" // resultado = "96"
```

Esto puede resultar ilógico, pero sí que tiene su lógica. La expresión se evalúa de izquierda a derecha. La primera operación funciona correctamente devolviendo el valor de 9 pero al intentar sumarle una cadena de texto "6" JavaScript lo que hace es convertir ese número a una cadena de texto y se lo concatenará al comienzo del "6".

Para convertir cadenas a números dispones de las funciones: `parseInt()` y `parseFloat()`.

Por ejemplo:

```
parseInt("34") // resultado = 34
parseInt("89.76") // resultado = 89
```

parseFloat devolverá un entero o un número real según el caso:

```
parseFloat("34") // resultado = 34
parseFloat("89.76") // resultado = 89.76
4 + 5 + parseInt("6") // resultado = 15
```

Si lo que deseas es realizar la conversión de números a cadenas, es mucho más sencillo, ya que simplemente tendrás que concatenar una cadena vacía al principio, y de esta forma el número será convertido a su cadena equivalente:

```
("" + 3400) // resultado = "3400"
("" + 3400).length // resultado = 4
```

En el segundo ejemplo podemos ver la gran potencia de la evaluación de expresiones. Los paréntesis fuerzan la conversión del número a una cadena. Una cadena de texto en JavaScript tiene una propiedad asociada con ella que es la longitud (length), la cual te devolverá en este caso el número 4, indicando que hay 4 caracteres en esa cadena "3400". La longitud de una cadena es un número, no una cadena.

4.-Operadores en JavaScript.

Los operadores nos permiten realizar cálculos o acciones sobre las variables o datos que maneja un script, devolviendo otro valor como resultado de dicha operación. JavaScript es un lenguaje rico en operadores: símbolos y palabras que realizan operaciones sobre uno o varios valores, para obtener un nuevo valor.

Cualquier valor sobre el cual se realiza una acción (indicada por el operador), se denomina un operando. Una **expresión** puede contener un operando y un

operador (denominado operador unario), como por ejemplo en `b++`, o bien dos operandos, separados por un operador (denominado operador binario), como por ejemplo en `a + b`.

Categorías de operadores en JavaScript	
Tipo	**Qué realizan**
Comparación	Comparan los valores de 2 operandos, devolviendo un resultado de true o false (se usan extensivamente en sentencias condicionales como `if... else` y en instrucciones `loop`). `==  !=  ===  !==  >  >=  <  <=`
Aritméticos	Unen dos operandos para producir un único valor que es el resultado de una operación aritmética u otra operación sobre ambos operandos `+ - * / % ++ - - +valor -valor`
Asignación	Asigna el valor a la derecha de la expresión a la variable que está a la izquierda. `= += -= *= /= %= <<= >= >>= >>>= &= |= ^= []`
Boolean	Realizan operaciones booleanas aritméticas sobre uno o dos operandos booleanos. `&&  ||  !`
Bit a bit	Realizan operaciones aritméticas o de desplazamiento de columna en las representaciones binarias de dos operandos. `&  |  ^  ~  <<  >>  >>>`
Objeto	Antes de que tengamos que invocar al objeto y sus propiedades o métodos. `. [] () delete in instanceOf new this`
Misceláneos	Operadores que tienen un comportamiento especial. `, ?: typeof void`

4.1.- Operadores de comparación.

Sirven para realizar expresiones condicionales todo lo complejas que deseemos. Estas expresiones se utilizan para tomar decisiones en función de la comparación de varios elementos, por ejemplo, si un número es mayor que otro o si son iguales.

Los operadores de comparación de JavaScript son.

Operadores de comparación en JavaScript			
Sintaxis	**Nombre**	**Tipo de comandos**	**Resultado**
==	Igualdad	Todos	Boolean
!=	Distinto	Todos	Boolean
===	Igualdad estricta	Todos	Boolean
!==	Desigualdad estricta	Todos	Boolean
>	Mayor que	Todos	Boolean
>=	Mayor o igual que	Todos	Boolean
<	Menor	Todos	Boolean
<=	Menor o igual que	Todos	Boolean

En valores numéricos, los resultados serían los mismos que obtendríamos con cálculos algebraicos.

Por ejemplo:

```
30 == 30 // true
30 == 30.0 // true
5 != 8 // true
9 > 13 // false
7.29 <= 7.28 // false
```

También podríamos comparar cadenas a este nivel:

```
"Marta" == "Marta" // true
"Marta" == "marta" // false
"Marta" > "marta" // false
"Mark" < "Marta" // true
```

Para poder comparar cadenas, JavaScript convierte cada carácter de la cadena de un string, en su correspondiente valor ASCII. Cada letra, comenzando con la primera del operando de la izquierda, se compara con su correspondiente letra en el operando de la derecha. Los valores ASCII de las letras mayúsculas, son más pequeños que sus correspondientes en minúscula, por esa razón "Marta" no es mayor que "marta". En JavaScript hay que tener muy en cuenta la sensibilidad a mayúsculas y minúsculas.

Si por ejemplo comparamos un número con su cadena correspondiente:

```
"123" == 123 // true
```

JavaScript cuando realiza esta comparación, convierte la cadena en su número correspondiente y luego realiza la comparación. También dispones de otra opción, que consiste en convertir mediante las funciones `parseInt()` o `parseFloat()` el operando correspondiente:

```
parseInt("123") == 123 // true
```

Los operadores `===` y `!==` comparan tanto el dato como el tipo de dato. El operador `===` sólo devolverá true, cuando los dos operandos son del mismo tipo de datos (por ejemplo, ambos son números) y tienen el mismo valor.

4.2.- Operadores aritméticos.

Son los utilizados para la realización de operaciones matemáticas simples como la suma, resta o multiplicación. En JavaScript son los siguientes:

Operadores aritméticos en JavaScript			
Sintaxis	**Nombre**	**Tipo de comandos**	**Resultado**
+	Mas	Integer, float, string	Integer, float, string
-	Menos	Integer, float	Integer, float
*	Multiplicación	Integer, float	Integer, float
/	División	Integer, float	Integer, float
%	Modulo	Integer, float	Integer, float
++	Incremento	Integer, float	Integer, float
--	Decremento	Integer, float	Integer, float

+valor	Positivo	Integer, float	Integer, float
-valor	negativo	Integer, float	Integer, float

Veamos algunos ejemplos:

```
var a = 10; // Inicializamos a al valor 10
var z = 0; // Inicializamos z al valor 0
z = a; // a es igual a 10, por lo tanto, z es igual a 10.
z = ++a; // el valor de a se incrementa justo antes de ser
asignado a z, por lo que a es 11 y z valdrá 11.
z = a++; // se asigna el valor de a (11) a z y luego se incrementa
el valor de a (pasa a ser 12).
z = a++; // a vale 12 antes de la asignación, por lo que z es
igual a 12; una vez
hecha la asignación a valdrá 13.
```

Otros ejemplos:

```
var x = 2;
var y = 8;
var z = -x; // z es igual a -2, pero x sigue siendo igual a 2.
z = - (x + y); // z es igual a -10, x es igual a 2 e y es igual a
8.
   z = -x + y; // z es igual a 6, pero x sigue siendo igual a 2 e y
   igual a 8.
```

4.3.- Operadores de asignación.

Sirven para asignar valores a las variables. En JavaScript son los siguientes:

Operadores de asignación en JavaScript			
Sintaxis	**Nombre**	**Ejemplo**	**Significado**
=	Asignación	x = y	x = y
+=	Sumar su valor	x += y	x = x + y
-=	Restar su valor	x -= y	x = x - y
*=	Multiplicar su valor	x *= y	x = x * y
/=	Dividir su valor	x /= y	x = x / y
%=	Módulo de su valor	x %= y	x = x % y
<<=	Desplazar bits a la izquierda	x <<= y	x = x <<y
>=	Desplazar bits a la derecha	x >= y	x = x > y
>>=	Desplazar bits a la dcha rellenando con 0	x >>= y	x = x >> y
>>>=	Desplazar bits a la derecha	x >>>= y	x = x >>> y
&=	Operación AND bit a bit	x &= y	x = x & y
|=	Operación OR bit a bit	x |= y	x = x | y
^=	Operación XOR bit a bit	x ^= y	x = x ^y
[]=	Desestructurando asignación	[a,b]=[c,d]	a=c, b=d

4.4.- Operadores booleanos.

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

Operandos booleanos en JavaScript			
Sintaxis	Nombre	Operandos	Resultados
&&	AND	Boolean	Boolean
||	OR	Boolean	Boolean
!	NOT	Boolean	Boolean

Ejemplos:

```
!true // resultado = false
!(10 > 5) // resultado = false
!(10 < 5) // resultado = true
!("gato" == "pato") // resultado = true
5 > 1 && 50 > 10 // resultado = true
5 > 1 && 50 < 10 // resultado = false
5 < 1 && 50 > 10 // resultado = false
5 < 1 && 50 < 10 // resultado = false
```

Tabla de valores de verdad del operador AND			
Operador izq	Operador AND	Operador dcho	Resultados
True	&&	True	True
True	&&	False	False
False	&&	True	False
False	&&	False	False

Tabla de valores de verdad del operador OR			
Operador izq	Operador OR	Operador dcho	Resultados
True	||	True	True
True	||	False	True
False	||	True	True
False	||	False	False

Ejemplos:

```
5 > 1 || 50 > 10 // resultado = true
5 > 1 || 50 < 10 // resultado = true
5 < 1 || 50 > 10 // resultado = true
5 < 1 || 50 < 10 // resultado = false
```

4.5.- Operadores bit a bit.

Para los programadores de scripts, las operaciones bit a bit suelen ser un tema avanzado. A menos que tú tengas que gestionar procesos externos en aplicaciones del lado del servidor, o la conexión con applets de Java, es raro que tengas que usar este tipo de operadores.

Los operandos numéricos, pueden aparecer en JavaScript en cualquiera de los tres formatos posibles (decimal, octal o hexadecimal). Tan pronto como el operador tenga un operando, su valor se convertirá a representación binaria (32 bits de longitud). Las tres primeras operaciones binarias bit a bit que podemos realizar son AND, OR y XOR y los resultados de comparar bit a bit serán:

Bit a bit AND: 1 si ambos dígitos son 1.

Bit a bit OR: 1 si cualquiera de los dos dígitos es 1.

Bit a bit XOR: 1 si sólo un dígito es 1.

Operadores bit a bit en JavaScript			
Sintaxis	**Nombre**	**Operando izq**	**Operando dcho**
&	Desplazamiento AND	Valor integer	Valor integer
|	Desplazamiento OR	Valor integer	Valor integer
^	Desplazamiento XOR	Valor integer	Valor integer
~	Desplazamiento NOT	ninguno	Valor integer
<<	Desplazamiento a la izquierda	Valor integer	Cantidad a desplazar
>>	Desplazamiento a la derecha	Valor integer	Cantidad a desplazar
>>>	Desplazamiento derecha rellenando con 0.	Valor integer	Cantidad a desplazar

Por ejemplo:

```
4 << 2 // resultado = 16
```

La razón de este resultado es que el número decimal **4** en binario es `00000100`. El operador `<<` indica a JavaScript que desplace todos los dígitos dos lugares hacia la izquierda, dando como resultado en binario `00010000`, que convertido a decimal te dará el valor **16**.

4.6.- Operadores de objeto.

Estos operadores se relacionan directamente con objetos y tipos de datos. La mayor parte de ellos fueron implementados a partir de las primeras versiones de JavaScript, por lo que nos podemos encontrar con algún tipo de incompatibilidad con navegadores.

. (punto)

El operador punto, indica que el objeto a su izquierda tiene o contiene el recurso a su derecha, como por ejemplo: `objeto.propiedad` y `objeto.metodo()`

Ejemplo con un objeto nativo de JavaScript:

```
var s = new String('pepe');
var longitud = s.length;
var pos = s.index0f("fa");            // resultado: pos = 2
```

[] (corchetes)

El operador punto, indica que el objeto a su izquierda tiene o contiene el

Por ejemplo cuando creamos un array: `var a = ["Santiago","Coruña","Lugo"];`

Enumerar un elemento de un array: `a[1] = "Coruña";`

Enumerar una propiedad de un objeto: `a["color"] = "azul";`

Delete (para eliminar un elemento de una colección)

Por ejemplo, si consideramos

```
var oceanos = new Array ("Atlántico", "Pacifico", "Índico",
"Artico);
```

podríamos hacer:

```
delete oceanos[2];
```

Esto eliminaría el tercer elemento del array ("Indico"), pero la longitud del array no cambiaría. Si intentamos referenciar esa posición oceanos[2] obtendríamos undefined.

In (para inspeccionar métodos o propiedades de un objeto)

El operando de la izquierda del operador, es una cadena referente a la propiedad o método (simplemente el nombre del método sin parámetros); el operando a la derecha del operador, es el objeto que estamos inspeccionando. Si el objeto conoce la propiedad o método, la expresión devolverá `true`.

Ejemplo: `"write" in document`

O también `"defaultView" in document`

instanceof (para comprobar si un objeto es una instancia de un objeto nativo de JavaScript)

Ejemplo:

```
a = new Array(1,2,3);
```

```
a instanceof Array; // devolverá true
```

`new (para acceder a los constructores de objetos incorporados en el núcleo de JavaScript`

Ejemplo:

```
var hoy = new Date();
// creará el objeto hoy de tipo Date() empleando el constructor por defecto de dicho objeto. this (para hacer referencia al propio objeto en el que estamos localizados)
```

Ejemplo:

```
nombre.onchange =validateInput;
function validateInput (evt)
{
    Var valorDeInput = this.value;
    //Este this hace referencia al objeto nombre que estamos
       validddndo
}
```

4.6.- Operadores misceláneos.

El operador coma,

Este operador, indica una serie de expresiones que van a ser evaluadas en secuencia, de izquierda a derecha. La mayor parte de las veces, este operador se usa para combinar múltiples declaraciones e inicializaciones de variables en una única línea.

Ejemplo:

```
var nombre, direccion, apellidos, edad;
```

Otra situación en la que podemos usar este operador coma, es dentro de la expresión `loop`. En el siguiente ejemplo inicializamos dos variables de tipo contador, y las incrementamos en diferentes porcentajes. Cuando comienza el bucle, ambas variables se inicializan a 0 y a cada paso del bucle una de ellas se incrementa en 1, mientras que la otra se incrementa en 10.

```
for (var i=0, j=0 ; i < 125; i++, j+10)
  {
    // más instrucciones aquí dentro
   }
```

Nota: no confundir la coma`,` con el delimitador de parámetros "`;`" en la instrucción `for`.

?: (operador condicional)

Este operador condicional es la forma reducida de la expresión `if … else`

La sintaxis formal para este operador condicional es:

`condicion ? expresión si se cumple la condición: expresión si no se cumple;`

Si usamos esta expresión con un operador de asignación:

```
var = condicion ? expresión si se cumple la condición: expresión si no se cumple;
```

Ejemplo:

```
var a,b;
a = 3; b = 5;
var h = a > b ? a : b; // a h se le asignará el valor 5;
```

typeof (devuelve el tipo de valor de una variable o expresión)

Este operador unario se usa para identificar cuando una variable o expresión es de alguno de los siguientes tipos: `number`, `string`, `boolean`, `object`, `function` o `undefined`.

Ejemplo:

```
if (typeof miVariable == "number")
{
    MiVariable = parseInt(miVariable);
}
```

5.-Condiciones y bucles.

Los programas pueden tomar decisiones y podemos lograr que un script repita un bloque de instrucciones las veces que se quiera. Para ello utilizaremos las estructuras de control, las cuales, nos permitirán controlar el flujo de nuestros programas.

En los programas generalmente se necesita hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir las mismas líneas de código una y otra vez. Para realizar cosas más complejas en nuestros scripts se utilizan las estructuras de control.

5.1.- Estructuras de control.

En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bucles de ejecución, se denominan "Estructuras de Control". Una estructura de control, dirige el flujo de ejecución a través de una secuencia de instrucciones, basadas en decisiones simples y en otros factores.

Una parte muy importante de una estructura de control es la "condición". Cada condición es una expresión que se evalúa a `true` o false.

En JavaScript tenemos varias estructuras de control, para las diferentes situaciones que te puedas encontrar durante la programación. Tres de las estructuras de control más comunes son: construcciones `if`, construcciones `if...else` y los `bucles`.

Construcción *if*.

La decisión más simple que podemos tomar en un programa, es la de seguir una rama determinada si una determinada condición es `true`. IF es una estructura de control utilizada para tomar decisiones. Es un condicional que sirve para realizar unas u otras operaciones en función de una expresión. Funciona de la siguiente manera, primero se evalúa una expresión, si da resultado positivo se realizan las acciones relacionadas con el caso positivo.

Sintaxis:

```
if (condición) // entre paréntesis irá la condición que se
                  evaluará a true o false.
{
         // instrucciones a ejecutar si se cumple la condición
}
```

Ejemplo:

```
if (miEdad >30)
{
   alert("Ya eres una persona adulta");
}
```

Construcción *if … else*.

En este tipo de construcción, podemos gestionar que haremos cuando se cumpla y cuando no se cumpla una determinada condición.

Sintaxis:

```
if (condición) // entre paréntesis irá la condición que se
                  evaluará a true o false.
{
     // instrucciones a ejecutar si se cumple la condición
}
else
{
     // instrucciones a ejecutar si no se cumple la condición
}
```

Ejemplo:

```
if (miEdad >30)
{
    alert("Ya eres una persona adulta.");
}
else
{
    alert("Eres una persona joven.");
}
```

Construcción *if anidadas*.

Para hacer estructuras condicionales más complejas podemos anidar sentencias IF, es decir, colocar estructuras IF dentro de otras estructuras IF. Con un solo IF podemos evaluar y realizar una acción u otra según dos posibilidades, pero si tenemos más posibilidades que evaluar debemos anidar IFs para crear el flujo de código necesario para decidir correctamente.

Ejemplo:

```
var numero1=23
var numero2=63
if (numero1 == numero2)
 {
   document.write("Los dos números son iguales")
 }else
  {
    if (numero1 > numero2)
     {
      document.write("El primer número es mayor que el segundo")
     }else
       {
      document.write("El primer número es menor que el segundo")
       }
  }
```

Estructura *switch*.

La estructura de control switch de Javascript es utilizada para tomar decisiones en función de distintos estados o valores de una variable.

Sintaxis:

```
switch (expresión)
 {
  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
}
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como deseemos. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra break es opcional, pero si no la ponemos a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en nuestro esquema anterior no hubiese ningún break y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y default.

También es opcional la opción default u opción por defecto.

5.2.- Bucles.

En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bucles de ejecución, se denominan "Estructuras de Control". Una estructura

Los bucles son estructuras repetitivas, que se ejecutarán un número de veces fijado expresamente, o que dependerá de si se cumple una determinada condición.

Bucle for.

Se utiliza para repetir las instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute.

Sintaxis:

```
for (expresión inicial; condición; incremento)
{
    // Instrucciones a ejecutar dentro del bucle.
}
```

El bucle `FOR` tiene tres partes incluidas entre los paréntesis, que nos sirven para definir cómo deseamos que se realicen las repeticiones. La primera parte es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.

La segunda parte es la condición, que se evaluará cada vez que comience una iteración del bucle. Contiene una expresión para decidir cuándo se ha de detener el bucle, o, mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.

Por último, tenemos la actualización, que 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.

Después del for se colocan las sentencias que queremos que se ejecuten en cada iteración, acotadas entre llaves.

Como se puede comprobar, **este bucle es muy potente, ya que en una sola línea podemos indicar muchas cosas distintas y muy variadas**, lo que permite una rápida configuración del bucle y una versatilidad enorme. Por ejemplo, si queremos escribir los números del 1 al 1.000 de dos en dos se escribirá el siguiente bucle.

Ejemplo:

```
for (var i=0; i<=100; i+=2)
{
   document.write(i);
   docuemnt.write("<br>");
}
```

Bucle while().

Este tipo de bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo de comprender que el bucle `FOR`, ya que no incorpora en la misma línea la inicialización de las variables, su condición para seguir ejecutándose y su actualización. Sólo se indica la condición que se tiene que cumplir para que se realice una iteración o repetición.

Sintaxis:

```
while (condición)
{
    // Instrucciones a ejecutar dentro del bucle.
}
```

Ejemplo:

```
var color = "";
while (color !="rojo")
{
    Color = prompt("Dame un color, rojo para finalizar", "");
}
```

Bucle do … while().

El tipo de bucle `do...while` es la última de las estructuras para implementar repeticiones de las que dispone JavaScript, y es una variación del bucle `while()`. Se utiliza generalmente, cuando no sabemos el número de veces que se habrá de ejecutar el bucle. Es prácticamente igual que el bucle `while()`, con la diferencia, de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

Sintaxis:

```
do {
      // Instrucciones a ejecutar dentro del bucle.
}while (condición);
```

Ejemplo:

```
var a = 1;
do{
    alert("El valor de a es: "+a); // Mostrará esta alerta 2
      veces.
    a++;
}while (a<3);
```

5.3.- Break y continue.

Estas son dos instrucciones que aumentan el control de los bucles en JavaScript. Sirven para parar y continuar con la siguiente iteración del bucle respectivamente.

Son dos instrucciones que se pueden usar en de las distintas estructuras de control y principalmente en los bucles, que sirven para salir del bucle o continuar.

- break: significa detener la ejecución de un bucle y salirse de él.
- continue: sirve para detener la iteración actual y volver al principio del bucle para realizar otra iteración, si corresponde.

Break.

Se detiene un bucle utilizando la palabra break. Detener un bucle significa salirse de él y dejarlo todo como está para continuar con el flujo del programa inmediatamente después del bucle.

```
for (i=0;i<10;i++){
document.write (i)
escribe = prompt("dime si continuo preguntando...", "si")
if (escribe == "no")
break
}
```

Este ejemplo escribe los números del 0 al 9 y en cada iteración del bucle pregunta al usuario si desea continuar. Si el usuario dice cualquier cosa continua, excepto cuando dice "no", situación en la cual se sale del bucle y deja la cuenta por donde se había quedado.

Continue.

Sirve para volver al principio del bucle en cualquier momento, sin ejecutar las líneas que haya por debajo de la palabra continue.

```
var i=0
while (i<7)
{
   incrementar = prompt("La cuenta está en " + i + ", dime si
      incremento", "si")
   if (incrementar == "no")
   continue
   i++
}
```

Este ejemplo, en condiciones normales contaría hasta desde i=0 hasta i=7, pero cada vez que se ejecuta el bucle pregunta al usuario si desea incrementar la variable o no. Si introduce "no" se ejecuta la sentencia continue, con lo que se vuelve al principio del bucle sin llegar a incrementar en 1 la variable i, ya que se ignorarían las sentencia que haya por debajo del continue.