Desarrollo Entorno Cliente

# 2.A.- Estructura del lenguaje JavaScript.

## 1.- Fundamentos de JavaScript

El lenguaje de programación objeto de este módulo es JavaScript.

JScript es la implementación particular de Microsoft, problemas de incompatibilidades con múltiples navegadores. W3C diseño el DOM para evitar esto. JScript se ha integrado en la plataforma de desarrollo .NET de Microsoft.

A partir de 1997 se rigen por un estándar ECMA. En el documento **ECMA**-**262** es dónde se detallan dichas especificaciones. JavaScript y JScript son compatibles con el estándar ECMA-262.

JavaScript sintaxis similar al lenguaje C. No tiene relación con Java ya que tienen semánticas y propósitos diferentes.

JavaScript fue desarrollado por **Brenda Eich**, lo llamo Mocha >> LiveScript >> JavaScript.

JS es uno de los lenguajes más extendidos y usados en la Web tanto en el desarrollo cliente como servidor que permiten desarrollar aplicaciones altamente acopladas con el cliente lo que quiere a éstas un altísimo grado de interactividad.

## 2.- Comentarios en el código.

A la hora de programar en cualquier lenguaje es muy importante incluir comentarios en el código para describir aspectos y facilitar el trabajo.

Los comentarios son sentencias que el intérprete de JavaScript ignora. Permite a los desarrolladores dejar notas cómo funcionan las cosas en sus programas.

Muy recomendable el que se documente el código siempre con límites.

JavaScript permite dos estilos de comentarios.

**//** este es un comentario de una línea

Para comentarios más largos **/\* \*/**

Funcionalidad que también tienen los comentarios:

* Desactivar el código para que deje de ejecutarlo sin tener que borrar nada.
* Dejar información de contacto para que otros programadores contacten contigo.

## 3.- Constantes y variables

En JavaScript a estos símbolos se les puede asignar un valor una única vez (constantes) o más de una (variables).

No es necesario declarar las variables antes de asignarles un valor.

Palabra reservada **const**.

const cajas = 3;

Dos palabras reservadas para declarar variables **var** y **let**:

**Var**, permite declarar la variable en su contexto de ejecución.

**Let**, permite declarar la variable limitado a su declaración al bloque, declaración o expresión en el que se va a utilizar.

Las variables se pueden declarar y luego asignar o hacer la declaración y asignación junta.

Var edad = 38;

Se puede omitir la palabra **var** sin embargo se recomiendo su uso.

Var altura, peso, edad;

Una variable de JavaScript podrá almacenar diferentes tipos de valores. No tienen tipo.

Dar nombres a las variables:

* no podremos usar palabras reservadas.
* ni símbolos de puntuación en el medio de la variable
* ni la variable podrá contener espacios en blanco.
* Se han de construir con caracteres alfanuméricos y el carácter subrayado (\_).
* No podremos utilizar caracteres raros como el signo +, un espacio, %, $, etc.
* En los nombres de variable y estos nombres no podrán comenzar con un número.
* Dos palabras se separarán con el símbolo “\_” o mayúsculas.

## 4.- Tipos de datos.

Tres tipos de datos:

* Booleano
* Numérico
* Cadena de caracteres

Palabra reservada **null** permite asignar un valor a una variable cuando todavía se desconoce su tipo o tiene un valor inválido.

El valor **undefined** se puede asignar a una variable cuando todavía no conocemos su valor.

## 4.1.- Conversiones de tipos de datos.

Aunque los tipos de datos en JS son muy sencillos, a veces las operaciones no se realizan correctamente y esto es debido a la conversión de tipos. JS lo realiza de la mejor forma y automática.

La expresión se evalúa de izquierda a derecha.

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

La conversión de números a cadenas, ya que simplemente se concatenará una cadena vacía al principio.

Los **paréntesis** fuerzan la conversión del número a una cadena.

JavaScript tiene una propiedad asociada con ella que es la longitud **length**.

## 5.- Operadores.

JS es un lenguaje rico en operadores: símbolos y palabras que realizan operaciones sobre uno o varios valores, para obtener un nuevo valor.

Cualquier valor se denomina operando. Y una expresión puede contener un operando y un operador o bien dos operandos separados por un operador.

Categorías de operadores en JavaScript:

* **Comparación**: compara los valores de 2 operandos, devolviendo un resultado de true o false. **== != === !== > >= < <=**
* **Aritméticos**: unen dos operandos para producir un único valor. **+ - \* / % ++ — +valor –valor**
* **Asignación**: asigna el valor a la derecha de la expresión a la variable que está a la izquierda. **= += -= \*= /= %= <<= >= >>= >>>= &= |= ^= []**
* **Booelan**: operaciones booleanas aritméticas sobre uno o dos operandos. **&& || !**
* **Bit a bit**: realizan operaciones aritméticas o de desplazamiento de columna. **& | ^ ~ << >> >>>**
* **Objeto**: ayudan a los scripts a evaluar la herencia y capacidades de un objeto. **[] . () delete in instanceOf new this**
* **Misceláneos**: operadores que tienen un comportamiento especial. **, ?: typeof void**

## 5.1 Operadores de comparación



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.

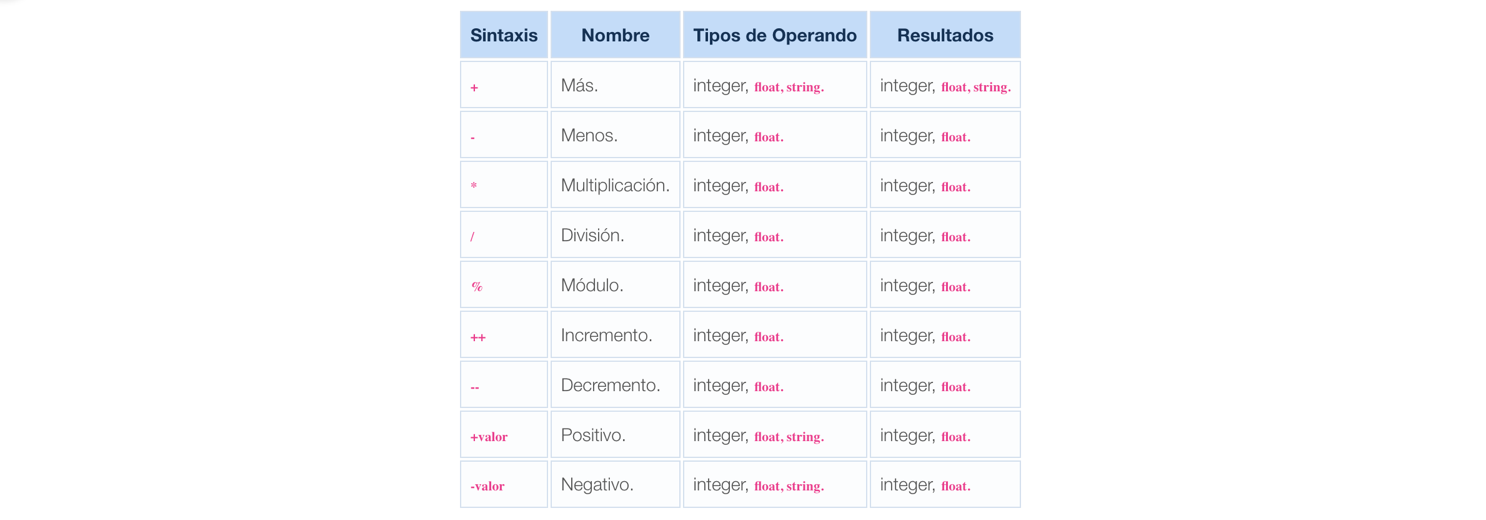
"123" == 123 // true

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.

## 5.3 Operadores aritméticos

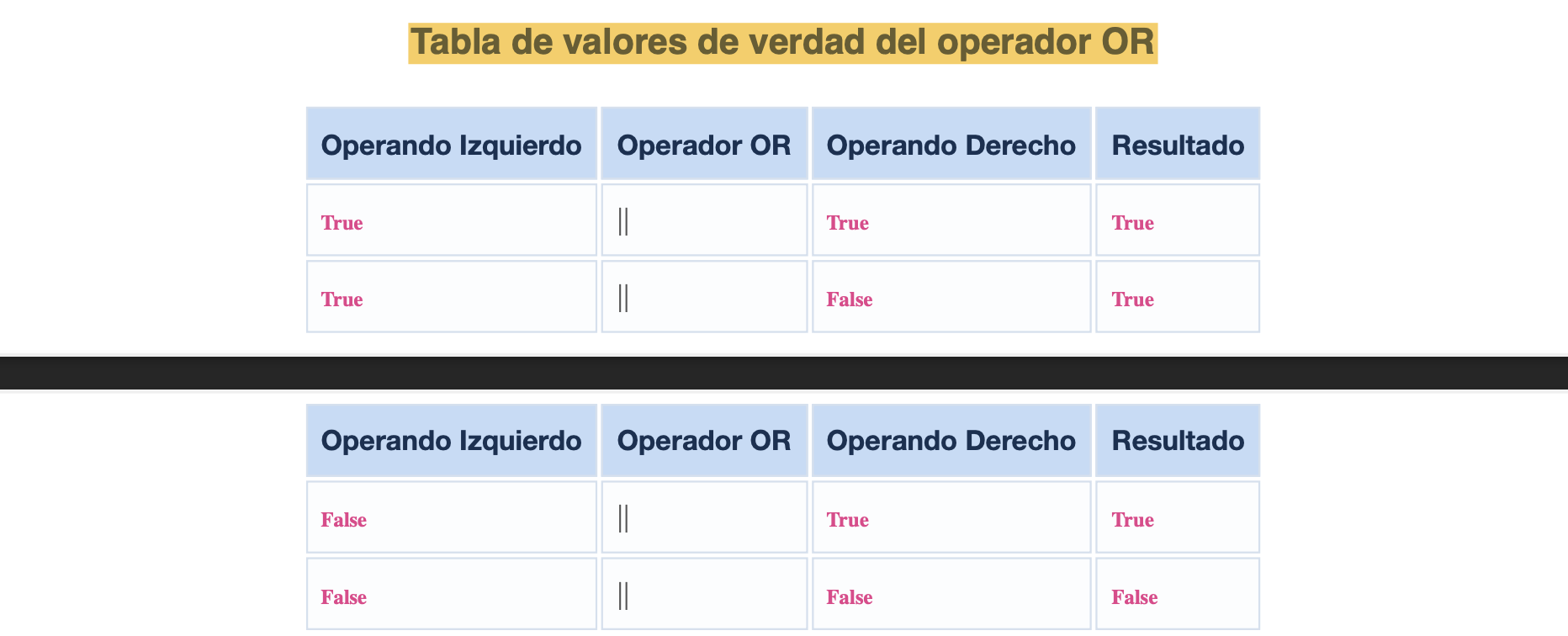
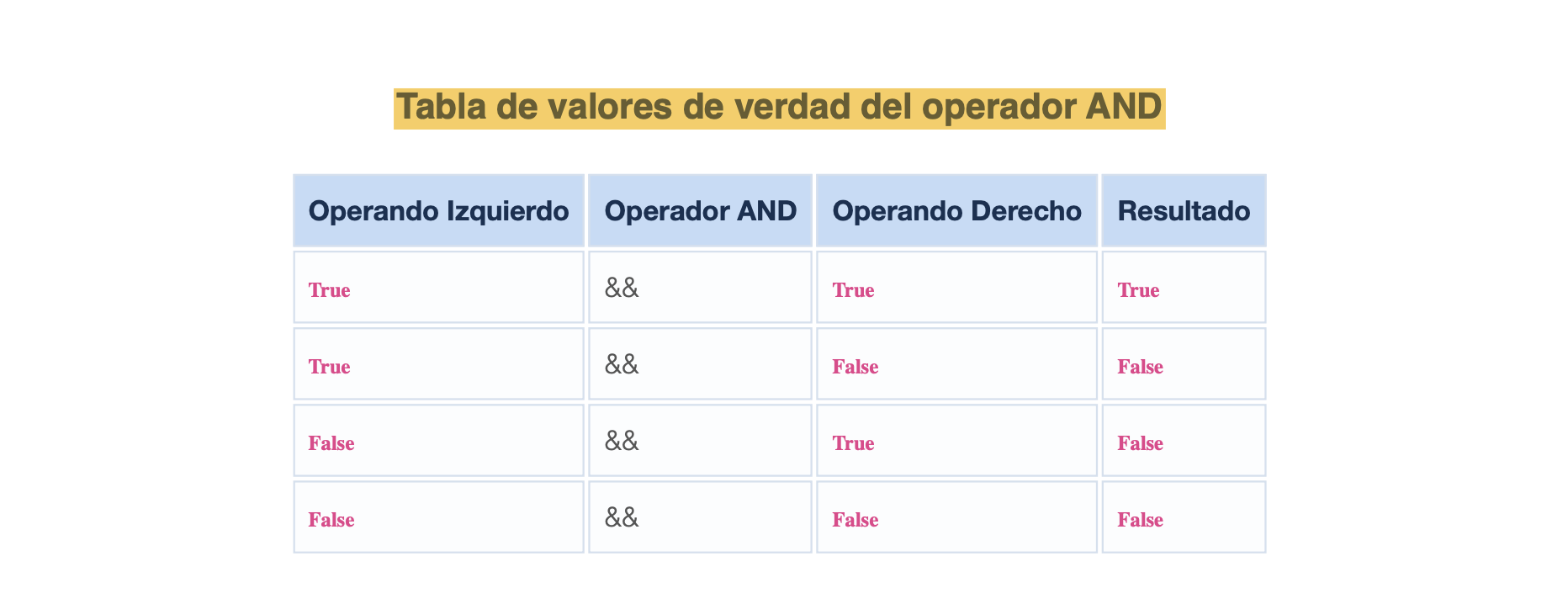
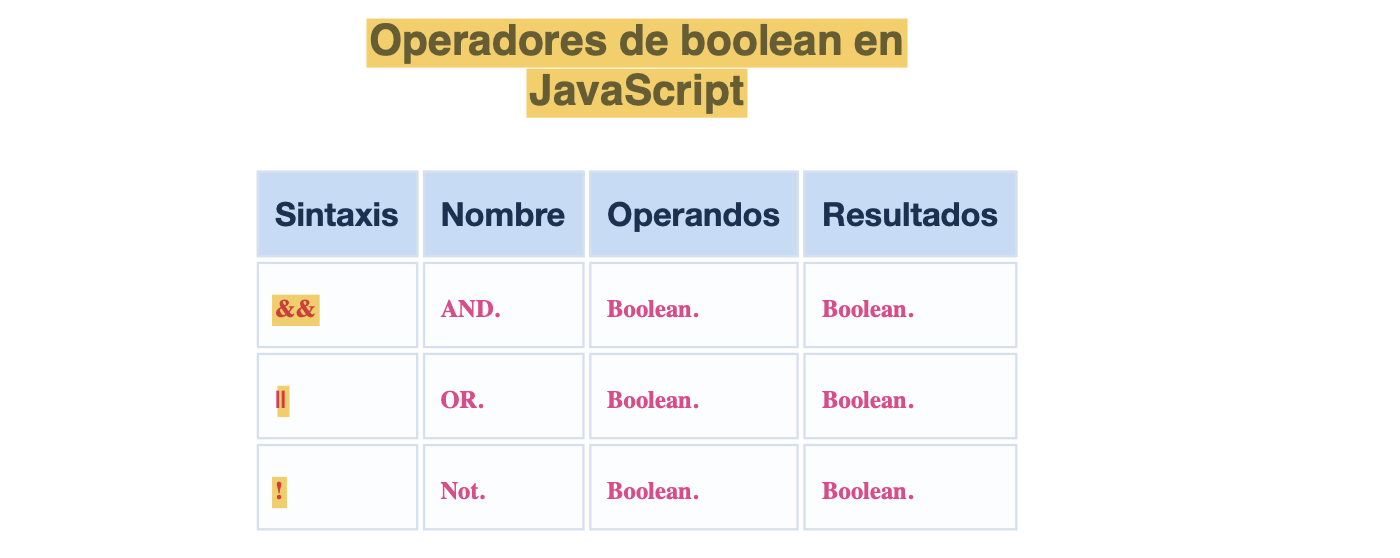


## 5.3 Operadores de asignación

* Asignación: **=**
* Sumar un valor: **+=**
* Substraer un valor: **-=**
* Multiplicar un valor: **\*=**
* Dividir un valor: **/=**
* Módulo de un valor: **%=**
* Desplazar bits a la izquierda: **<<=**
* Desplazar bits a la derecha: **>=**
* Desplazar bits a la derecha rellenando con 0: **>>=**
* Desplazar bits a la derecha: **>>>=**
* Operación AND bit a bit: **&=**
* Operación OR bit a bit: **!=**
* Operación XOR bit a bit: **^=**
* Desestructurando asignaciones: **[]=**

## 5.4.- Operadores booleanos

Los operadores booleanos juegan un gran papel, van a permitir evaluar expresiones, devolviendo como resultado true o false.



## 5.6.- Operadores de Objeto

Fueron implementados a partir de las primeras versiones de JS, por lo que puede haber algún tipo de incompatibilidad con navegadores antiguos.

**. (punto)** : operador punto, indica que el objeto a su izquierda tiene o contiene el recurso a su derecha.

* Objeto.propiedad, objeto.método()

**[] (corchetes)**: para enumerar miembros de un objeto.

* var a = [“nombre”,”apellido”]
* a[1] = “ciudad”;
* a[“color”] = “azul”;

**Delete** : para eliminar un elemento de una colección, pero la longitud del array no cambiaría.

* var oceanos = new Array (“Atlantico”, “Pacífico”, “Indico”, “Artico”);
* delete oceanos [2];

**In** : para inspeccionar métodos o propiedades de un objeto.

"write" in document o también "defaultView" in document

**Instanceof**: para comprobar si un objeto es una instancia de un objeto nativo de JavaScript.

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

a instanceof Array; // devolverá true.

**New**: para acceder a los constructores de objetos incorporados en el núcleo de JS.

var hoy = new Date();

**This**: para hacer referencia al propio objeto en el que estamos localizados.

function validateInput(evt) {

nombre.onchange = validateInput;

var valorDeInput = this.value;// Este this hace referencia al objeto nombre que estamos validando.

}

## 5.5.- Operadores bit a bit.

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:

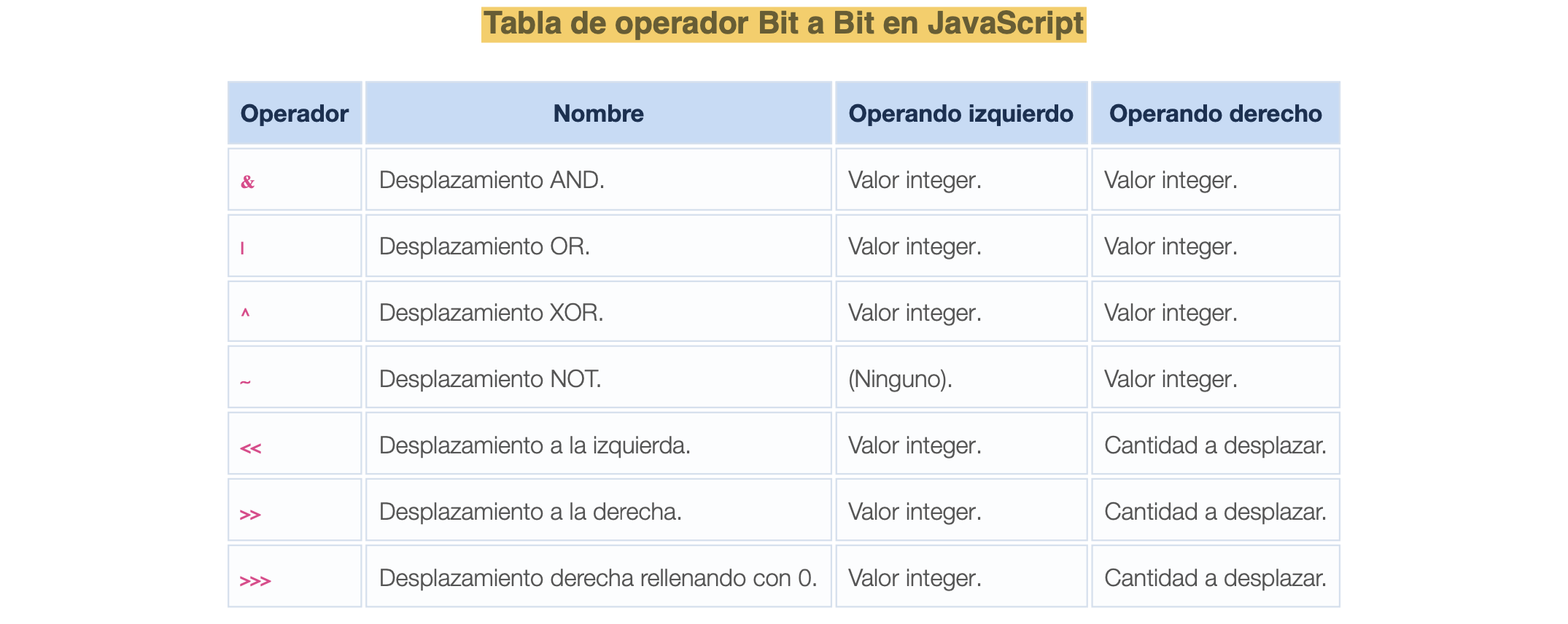
Bit a bit AND: 1 si ambos dígitos son 1.

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

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

4 << 2 // resultado = 16

00000100 — 00010000



## 5.7.- Operadores misceláneos

El operador coma **(,)** indica una serie de expresiones que van a ser evaluadas en secuencia, de izquierda a derecha.

Var nombre, direccion, apellidos, edad;

También se usa dentro de la expresión loop.

for (var i=0, j=0 ; i<125;i++, j+10){

//má instrucciones

}

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

**?:(operador condicional)**

Es la forma reducida de la expresión if...else.

*Condición ? Expresión si se cumple la condición : expresión si no se cumple*.

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)**

Operador unario, que sirve para identificar una variable o expresión de algunos de estos tipos: **number, string, boolean, object, function o undefined.**

if (typeof miVariable == "number") {

miVariable = parseInt(miVariable);

}

## 6.- Estructuras de control

En esta sección te mostraremos cómo los programas pueden tomar decisiones y cómo puedes lograr que un script repita un bloque de instrucciones las veces que quieras.

## 6.1.- Sentencias condicionales

En los lenguajes de programación, las sentencias condicionales permiten decidir acciones en función del resultado de evaluar condiciones o expresiones lo que nos permite dirigir el flujo de ejecución de nuestro programa.

JavaScript algunos valores que no son booleanos tienen implícito el valor false cuando se evalúan en una condición. 0 y “ “ tienen el valor de false.

En JavaScript tenemos varias sentencias condicionales para las diferentes situaciones que te puedas encontrar durante la programación.

**Construcción IF**

if (typeof miVariable == "number") {

miVariable = parseInt(miVariable);

}

**Construcción IF...ELSE**

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

}

**Construcción SWITCH**

Equivalente a **if...else** anidadas, sin embargo, es bastante más legible de cara a los desarrolladores. **Break** para romper el flujo.

switch (expresión) { // entre paréntesis escribimos la expresión a evaluar

case valor1: // instrucciones a ejecutar si la evaluación de la expresión coincide con valor1

break; // importante para romper el flujo

case valor2: // instrucciones a ejecutar si la evaluación de la expresión coincide con valor2

break; // importante para romper el flujo

........

case valorN: // instrucciones a ejecutar si la evaluación de la expresión coincide con valorN

break: // importante para romper el flujo default: // instrucciones a ejecutar si la evaluación de la expresión no coincide con ningún valor.

}

## 6.2.- Bucles

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**

Repite un bloque de instrucciones un número limitado de veces.

for (expresión inicial; condición; incremento) {

// Sentencias a ejecutar dentro del bucle.

}

**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.

while (condición) {

// Sentencias a ejecutar dentro del bucle.

}

**Bucle DO...WHILE()**

Este bucle es una variación del bucle while(). Se utiliza generalmente, cuando no sabemos el número de veces que se habrá de ejecutar el bucle.

do {

// Sentencias a ejecutar dentro del bucle.

} while (condición);