A yellow square containing the letters 'JS' in a large, bold, black sans-serif font.

Desarrollo web en entorno cliente

UD – 2

Manejo de la sintaxis del lenguaje

Contenidos



ÍNDICE

Introducción.....	3
Comentarios.....	3
Variables.....	4
<i>Tipos de variables.....</i>	<i>4</i>
<i>Arrays.....</i>	<i>5</i>
<i>Conversiones entre tipos.....</i>	<i>5</i>
Mostrar por pantalla y leer del teclado.....	6
<i>alert() , confirm(), prompt().....</i>	<i>6</i>
<i>Template Literals.....</i>	<i>6</i>
Operadores.....	7
<i>Operadores de asignación.....</i>	<i>7</i>
<i>Operadores aritméticos.....</i>	<i>8</i>
<i>Operadores de comparación.....</i>	<i>9</i>
<i>Operadores lógicos.....</i>	<i>10</i>
Estructuras de control y bucles.....	11
<i>IF y ELSE.....</i>	<i>11</i>
<i>Bucle FOR.....</i>	<i>12</i>
<i>Bucle WHILE.....</i>	<i>13</i>
<i>Bucle DO-WHILE.....</i>	<i>13</i>
<i>Instrucciones BREAK y CONTINUE.....</i>	<i>14</i>
Funciones.....	15
<i>Uso del array arguments.....</i>	<i>15</i>
<i>Uso de funciones anónimas.....</i>	<i>15</i>
<i>funciones lambda o funciones flecha.....</i>	<i>16</i>
Cambiar dinámicamente propiedades de objetos.....	17
Variables locales y globales.....	17

1. INTRODUCCIÓN

Javascript es un lenguaje de programación que permite ejecutar código en el cliente (nuestro navegador) ampliando la funcionalidad nuestros sitios web. Para añadir JavaScript se usa la etiqueta **<script>**. Este puede estar en cualquier lugar de la página. El código se ejecuta en el lugar donde se encuentra de forma secuencial a como el navegador lo va encontrando.

```
<SCRIPT LANGUAGE="JavaScript">

<!--
    Aquí va el código
    // Esto es un comentario en Javascript de una línea
-->

</SCRIPT>
```

2. COMENTARIOS

En Javascript los **comentarios** se pueden hacer con:

- `//` → Comentarios de una línea.
- `/* */` → Comentarios de varias líneas.

Desde Javascript es posible reescribir el código HTML de la página mediante la orden

document.write(texto)

Este ejemplo podría ser un pequeño hola mundo que al ejecutarse modificará el HTML de la página. Ejemplo **document.write(texto)** y comentario multilínea.

```
<SCRIPT LANGUAGE="JavaScript">

    document.write ("Hola mundo");

    /* Esto es un comentario en Javascript multilínea */

</SCRIPT>
```

Otra vía para mostrar información al usuario desde una ventana emergente, es el comando **alert(texto)**.

```
<SCRIPT LANGUAGE="JavaScript">

<!--
    alert("Hola mundo");
-->

</SCRIPT>
```

Hay una forma mucho más práctica y ordenada de usar código Javascript. Se pueden incluir uno o varios ficheros con código Javascript. Se puede incluir tantos como se desee. Un ejemplo de inclusión de ficheros.

```
<script type="text/javascript" src="rutaDelArchivo1.js"/>
<script type="text/javascript" src="rutaDelArchivo2.js"/>
<script type="text/javascript" src="rutaDelArchivo3.js"/>
```

3. VARIABLES

Las variables son elementos del lenguaje que permiten almacenar distintos valores en cada momento. Se puede almacenar un valor en una variable y consultar este valor posteriormente. También podemos modificar su contenido siempre que queramos.

Para declarar las variables en JavaScript se utiliza la palabra reservada **var**. A cada variable se le asigna un nombre, y opcionalmente, un valor inicial. Si no se trata de una función, la instrucción var es opcional, pero se recomienda utilizarla.

```
var ejemplo;  
var resultado;  
ejemplo= "Hola";  
resultado=3+7;
```

En JavaScript todas las variables y nombres de función son **sensibles a mayúsculas y minúsculas**. Esto significa que la capitalización importa. Escribe los nombres de las variables en JavaScript en camelCase. En **camelCase**, los nombres de variables de múltiples palabras tienen la primera palabra en minúsculas y la primera letra de cada palabra posterior en mayúsculas.

3.1 Tipos de variables

JavaScript proporciona ocho tipos de datos diferentes, los cuales son:

undefined, null, boolean, string, symbol, bigint, number, y object.

- **Números:** puede contener cualquier tipo de número real (0.3,1.7,2.9) o entero (5,3,-1).
- **Operadores lógicos o boolean:** puede contener uno de los siguientes valores: true, false, 1 y 0.
- **Cadenas de caracteres o String:** cualquier combinación de caracteres (letras, números, signos especiales y espacios). Las cadenas se delimitan mediante comillas dobles o simples ("Lolo","laO"). Para concatenar cadenas puede usarse el operador + ("Soy" + "el " + "AMO" formaría la cadena "Soy el AMO").
- **Undefined:** Valor no definido, es decir, la variable no tiene asignado un valor.

```
var edad=23, nueva_edad, incremento;  
var nombre="Rosa García";  
incremento=4;  
nueva_edad=edad+incremento;  
alert(nombre+ " dentro de "+incremento +" años tendrá "+ nueva_edad+"  
años");
```

Puedes utilizar la función **typeof** para consultar el tipo de la variable o dato:

```
console.log(typeof(5))
```

3.2 Arrays

Un array (también llamado vector o arreglo) es una variable que contiene diversos valores. Lo creamos con “new Array()” o inicializando los elementos. Podemos referenciar esos valores indicando su posición en el array. Los arrays poseen una propiedad llamada “**length**” que podemos utilizar para conocer el número de elementos del array.

```
// Array definido 1 a 1
var miVector=new Array();
miVector[0]=22;
miVector[1]=12;
miVector[2]=33;

//Array definido en una línea
var otroArray=[1,2,"Cancamusa"];

// Valores dentro del array
alert(miVector[1]);
alert(otroArray[2]);

// Valor completo del array
alert(miVector);

// Tamaño del array
alert(miVector.length);
```

3.3 Conversiones entre tipos

Javascript no define explícitamente el tipo de datos de sus variables. Según se almacenen, pueden ser cadenas (Entrecomillados), enteros (sin parte decimal) o decimales (con parte decimal).

Elementos como la función “prompt” para leer de teclado, leen los elementos siempre como cadena. Para estos casos y otros, merece la pena usar funciones de conversión de datos.

```
var num="100";           // Es una cadena
var num2="100.13";       // Es una cadena
var num3=11;             // Es un entero
var n=parseInt(num);      // Almacena un entero.
var n2=parseFloat(num);  // Almacena un decimal
var n3=num3.toString();  // Almacena una cadena
```

4. MOSTRAR POR PANTALLA Y LEER DEL TECLADO

4.1 `alert()`, `confirm()`, `prompt()`

El método **`alert()`** permite mostrar al usuario información literal o el contenido de variables en una ventana independiente. La ventana contendrá la información a mostrar y el botón aceptar.

```
alert("Bienvenido al módulo DWEC");
```

A través del método **`confirm()`** se activa un cuadro de diálogo que contiene los botones Aceptar y Cancelar.

- **True** → Se ha pulsado la opción aceptar (botón).
- **False** → Se ha pulsado la opción cancelar (botón).

Con ayuda de este método el usuario puede decidir sobre preguntas concretas e influir de ese modo directamente en la página.

```
var respuesta;  
respuesta=confirm("¿Eres bienvenido al módulo DWEC?");  
alert("Usted ha contestado que "+respuesta);
```

El método **`prompt()`** abre un cuadro de diálogo en pantalla en el que se pide al usuario que introduzca algún dato. Si se pulsa el botón Cancelar, el valor de devolución es false/null. Pulsando en Aceptar se obtiene el valor true y la cadena de caracteres introducida se guarda para su posterior procesamiento.

```
var provincia;  
provincia=prompt("Introduzca la provincia ", "Valencia");  
alert("Usted ha introducido la siguiente información "+provincia)
```

4.2 *Template Literals*

Desde ES2015 también podemos poner una cadena entre ``` (acento grave) y en ese caso podemos poner dentro variables y expresiones que serán evaluadas al ponerlas dentro de `${}`. También se respetan los saltos de línea, tabuladores, etc que haya dentro. Ejemplo:

```
> let edad = 25;  
< undefined  
> edad  
< 25  
> console.log(`El usuario tiene:  
  ${edad} años`)  
El usuario tiene:  
  25 años
```

Mostrará en la consola: → “El usuario tiene: 25 años ” (Con el salto de línea correspondiente)

5. OPERADORES

Combinando variables y valores, se pueden formular expresiones más complejas. Las expresiones son una parte esencial de los programas. Para formular expresiones se utilizan los operadores.

5.1 Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las variables. Alguno de ellos también incluyen operaciones.

Operador	Descripción
=	Asigna a la variable de la parte izquierda el valor de la parte derecha.
+=	Suma los operandos izquierdo y derecho y asigna el resultado al operando izquierdo.
-=	Resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo.
*=	Multiplica ambos operandos y asigna el resultado al operando izquierdo.
/=	Divide ambos operandos y asigna el resultado al operando izquierdo.
%=	Divide ambos operandos y asigna el resto al operando izquierdo.

```
var num1=3;
var num2=5;
num2+=num1;
num2-=num1;
num2*=num1;
num2/=num1;
num2%=num1;
```

5.2 Operadores aritméticos

Los operadores aritméticos se utilizan para hacer cálculos aritméticos.

Operador	Descripción
+	Suma.
-	Resta.
*	Multiplica.
/	Divide.
%	Calcula el resto de una división.

Además de estos operadores, también existen operadores aritméticos unitarios: incremento (++), disminución (--) y la negación unitaria (-). Los operadores de incremento y disminución pueden estar tanto delante como detrás de una variable. Estos operadores aumentan o disminuyen en 1, respectivamente, el valor de una variable.

La diferencia entre ambas posiciones reside en el momento en que se ejecuta la operación.

Operador (x=5)	Descripción
y= ++ x	Primero el incremento y después la asignación. y=6
y = x++	Primero la asignación y después el incremento. y=5
y = -- x	Primero el decremento y después la asignación. y=4
y = x--	Primero la asignación y después el decremento. y=5
y =-x	Se asigna a y el valor negativo de x, pero x no varía. y=-5

```
var num1=5, num2=8,resultado1, resultado2;
resultado1=((num1+num2)*200)/100;
resultado2=resultado1%3;
resultado1=++num1;
resultado2=num2++;
resultado1=- -num1;
resultado2=num2--;
```


5.3 Operadores de comparación

Para comparar dos valores entre sí, se utiliza el operador de comparación. Como valor de retorno se obtiene un valor lógico o booleano: *true* o *false*.

Operador	Descripción
<code>==</code>	<i>Devuelve el valor true cuando los dos operandos son iguales.</i>
<code>!=</code>	<i>Devuelve el valor true cuando los dos operandos son distintos.</i>
<code>></code>	<i>Devuelve el valor true cuando el operando de la izquierda es mayor que el de la derecha.</i>
<code><</code>	<i>Devuelve el valor true cuando el operando de la derecha es menor que el de la izquierda.</i>
<code>>=</code>	<i>Devuelve el valor true cuando el operando de la izquierda es mayor o igual que el de la derecha.</i>
<code><=</code>	<i>Devuelve el valor true cuando el operando de la derecha es menor o igual que el de la izquierda.</i>

```
var a=4;b=5;  
alert("El resultado de la expresión 'a==b' es igual a "+(a==b));  
alert("El resultado de la expresión 'a!=b' es igual a "+(a!=b));  
alert("El resultado de la expresión 'a>b' es igual a "+(a>b));  
alert("El resultado de la expresión 'a<b' es igual a "+(a<b));  
alert("El resultado de la expresión 'a>=b' es igual a "+(a>=b));  
alert("El resultado de la expresión 'a<=b' es igual a "+(a<=b));
```

5.4 Operadores lógicos

Los operadores lógicos se utilizan para el procesamiento de los valores booleanos. A su vez el valor que devuelven también es booleano: **true** o **false**.

Operador	Descripción
&&	<i>Y lógica. El valor de devolución es true cuando ambos operandos son verdaderos.</i>
 	<i>O lógica. El valor de devolución es true cuando alguno de los operandos es verdadero o lo son los dos.</i>
!	<i>No lógica. El valor de devolución es true cuando el valor es falso.</i>

Ejemplo: Se muestra el resultado de distintas operaciones realizadas con operadores lógicos. (En el ejemplo se usa directamente los valores true y false en lugar de variables).

```
alert("resultado 'false&&false' es igual a "+(false&&false));  
alert("resultado 'false&&true' es igual a "+(false&&true));  
alert("resultado 'true&&false' es igual a "+(true&&false));  
alert("resultado 'true&&true' es igual a "+(true&&true));  
alert("resultado 'false||false' es igual a "+(false||false));  
alert("resultado 'false||true' es igual a "+(false||true));  
alert("resultado 'true||false' es igual a "+(true||false));  
alert("resultado 'true||true' es igual a "+(true||true));  
alert("resultado '!false' es igual a "+(!false));
```

6. ESTRUCTURAS DE CONTROL Y BUCLES

6.1 IF y ELSE

Para controlar el flujo de información en los programas JavaScript existen una serie de estructuras condicionales y bucles que permiten alterar el orden secuencial de ejecución.

La instrucción **if** permite la ejecución de un bloque u otro de instrucciones en función de una condición.

Sintaxis:

```
if (condición) {  
    // bloque instrucciones, se ejecutan si la condición se cumple  
}  
else{  
    // bloque instrucciones, se ejecutan si la condición no se cumple  
}
```

Las llaves solo son obligatorias cuando haya varias instrucciones seguidas pertenecientes a la ramificación.

Si no pones llaves, el **if** se aplicará únicamente a la siguiente instrucción (*primera instrucción*).

Puede existir una instrucción **if** que no contenga la parte **else**. En este caso, se ejecutarían una serie de órdenes si se cumple la condición y si esto no es así, se continuaría con las órdenes que están a continuación del bloque **if**.

```
var edadAna, edadLuis;  
edadAna=parseInt(prompt("Introduce la edad de Ana",""));  
edadLuis=parseInt(prompt("Introduce la edad de Luis",""));  
  
if (edadAna > edadLuis){  
    alert("Ana es mayor que Luis.");  
    alert(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);  
}  
  
else{  
    alert("Ana es menor o de igual edad que Luis.");  
    alert(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);  
}
```

Para las condiciones ramificadas más complicadas, a menudo se utilizan las ramificaciones anidadas. En ellas se definen consultas **if** dentro de otras consultas **if**.

```
var edadAna, edadLuis;  
edadAna=parseInt(prompt("Introduce la edad de Ana",""));  
edadLuis=parseInt(prompt("Introduce la edad de Luis",""));  
  
if (edadAna > edadLuis){  
    alert("Ana es mayor que Luis.");  
}  
  
else{  
    if (edadAna < edadLuis){  
        alert("Ana es menor que Luis.");  
    } else{  
        alert("Ana tiene la misma edad que Luis.");  
    }  
}  
  
alert(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);
```

6.2 Bucle FOR

Cuando la ejecución de un programa llega a un bucle **for**, lo primero que hace es ejecutar la “*inicialización del índice*”, que solo se ejecuta un vez, a continuación analiza la “*condición de prueba*” y si esta se cumple ejecuta las instrucciones del bucle. Cuando finaliza la ejecución de las instrucciones del bucle se realiza la “*modificación del índice*” y las líneas de código que contiene el bucle y cuando estas finalizan, se retorna a la cabecera del bucle for y se realiza la después la Condición de prueba, si la condición se cumple se ejecutan las instrucciones y si no se cumple la ejecución continúa en las líneas de código que siguen al bucle.

Sintaxis

```
for (inicialización_indice; condición_prueba; Modificación_indice){  
    // Bloque de instrucciones...  
}
```

Ejemplo: números pares del 2 al 30

```
for (i=2; i<=30; i+=2) {  
    alert(i);  
}  
alert(" Ya se han escrito los números pares del 0 al 30");
```

Ejemplo: números pares del 0 al 30

```
for (i=30; i>=2; i-=2) {  
    alert(i);  
}  
alert("Ya se han escrito los números pares del 0 al 30");
```

Ejemplo: Escribe las potencias de 2 hasta 3000

```
aux=1;  
for (i=2; i<=3000; i*=2) {  
    alert("2 elevado a "+aux+" es igual a "+i);  
    aux++;  
}  
alert("Ya se han escrito las potencias de 2 menores de 3000");
```

6.3 Bucle WHILE

Con el bucle **while** se pueden ejecutar un grupo de instrucciones mientras se cumpla una condición. Si la condición nunca se cumple, entonces tampoco se ejecuta ninguna instrucción.

Si la condición se cumple siempre, nos veremos inmersos en el problema de los bucles infinitos, que pueden llegar a colapsar el navegador, o incluso el ordenador. Por esa razón es muy importante que la condición deba dejar de cumplirse en algún momento para evitar bucles infinitos.

```
while (condición){  
    // Bloque de instrucciones...  
}
```

Ejemplo : Escribe los números pares de 0 a 30

```
i=2;  
while (i<=30) {  
    alert (i);  
    i+=2;  
}  
alert("Ya se han mostrado los números pares del 0 al 30");
```

Ejemplo: Preguntar una clave hasta que se corresponda con una dada.

```
auxclave="";  
while (auxclave!="vivaY0"){  
    auxclave=prompt("introduce la clave ", "claveSecreta")  
}  
alert ("Has acertado la clave");
```

6.4 Bucle DO-WHILE

La diferencia del bucle **do-while** frente al bucle while reside en el momento en que se comprueba la condición:

- El bucle *do-while* no la comprueba hasta el final, es decir, después del cuerpo del bucle, lo que significa que el bucle *do-while* se recorrerá, una vez, como mínimo, aunque no se cumpla la condición.

```
do {  
    // Bloque de instrucciones...  
} while (condición)
```

Ejemplo: Preguntar por una clave hasta que se introduzca la correcta

```
do {  
    auxclave=prompt("introduce la clave ", "vivaYo")  
} while (auxclave!="EstaeslaclaveJEJEJE")  
  
alert ("Has acertado la clave");
```

6.5 Instrucciones **BREAK** y **CONTINUE**

En los bucles for, while y do-while se pueden utilizar las instrucciones **break** y **continue** para modificar el comportamiento del bucle.

- **Break** → Hace que el bucle se interrumpa inmediatamente, aun cuando no se haya ejecutado todavía el bucle completo. Al llegar la instrucción, el programa se sigue desarrollando inmediatamente a continuación del final del bucle.

Ejemplo: *Pregunta por la clave y permitir tres respuestas incorrectas*

```
var auxclave=true , var numveces=0;
while (auxclave != "anonimo" && auxclave){
    auxclave=prompt("Introduce la clave ", "");
    numveces++;
    if (numveces == 3)
        break;
}
if (auxclave=="SuperClave") alert("La clave es correcta");
else alert("La clave no es correcta correcta");
```

- **Continue** → Hace retornar a la secuencia de ejecución a la cabecera del bucle, volviendo a ejecutar la condición o a incrementar los índices cuando sea un bucle for.

Ejemplo: *Presenta todos los números pares del 0 al 50 excepto los que sean múltiplos de 3*

```
var i;
for (i=2;i<=50;i+=2){
    if ((i%3)==0)
        continue;
    alert(i);
}
```

7. FUNCIONES

Se declaran con **function** y se les pasan los parámetros entre paréntesis. La función puede devolver un valor usando **return** (si no tiene return es como si devolviera undefined).

```
function nombrefuncion (parámetro1, parámetro2...){
  ...
  // Bloque de instrucciones
  ...
  //si la función devuelve algún valor añadimos:
  return valor;
}
```

Ejemplo: Funciones que devuelve la suma de dos valores que se pasan por parámetros y que escriben el nombre del profesor.

```
// Definiciones de las funciones
function suma (a,b){
  return a+b;
}
function profe (){
  alert ("El profesor es: Agustin Aguilera");
}

// Código que se ejecuta
var op1=5;op2=25;
var resultado;

// Llamadas a funciones
resultado=suma(op1,op2);
console.log (op1+"+"+op2+"="+resultado);
profe();
```

7.1 Uso del array arguments

También es posible acceder a los parámetros desde el array **arguments[]** si no sabemos cuántos recibiremos:

```
function suma () {
  var result = 0;
  for (var i=0; i<arguments.length; i++)
    result += arguments[i];
  return result;
}
```

7.2 Uso de funciones anónimas

Podemos definir una función sin darle un nombre. Dicha función puede asignarse a una variable, autoejecutarse o asignarse a un manejador de eventos. Ejemplo:

```
var suma = function (a,b){
  return a+b;
}
```

7.3 funciones lambda o funciones flecha

Para escribir la sintaxis de una función flecha, debemos tener en consideración:

- *Eliminamos la palabra function*
- *Ponemos el símbolo =>*
- *Si sólo tiene 1 parámetro podemos eliminar los paréntesis de los parámetros*
- *Si la función sólo tiene 1 línea podemos eliminar las {} y la palabra return*

Función Anónima	Función flecha
<pre>var suma = function (a,b){ return a+b; }</pre>	<pre>var suma = (a,b) => {return a+ b} var suma = (a,b) => a+ b</pre>

Un ejemplo de ejecución en la consola del navegador sería:

```
> var suma_arrow_1 = (a,b) => {return a+ b}  
< undefined  
> suma_arrow_1(1,2)  
< 3  
> var suma_arrow_2 = (a,b) => a+b;  
< undefined  
> suma_arrow_2(1,2)  
< 3
```


8. CAMBIAR DINÁMICAMENTE PROPIEDADES DE OBJETOS

Conociendo la id de algún objeto HTML existente en la página podemos cambiar sus propiedades. Cuando queremos aplicárselo a un objeto del documento utilizamos el ID y el método **document.getElementById()** para acceder al elemento.

Para cambiar una imagen con id 'matrix', modificamos la propiedad src :

```
document.getElementById('matrix').src = "mt05.jpg";
```

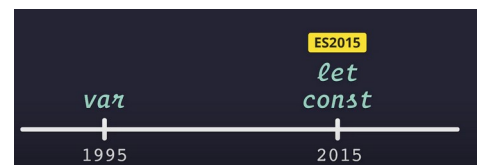
Esto se puede usar para cualquier propiedad existente en cualquier objeto HTML. La función Javascript genérica para cambiar una imagen sería:

```
function cambiarImagen (id,rutaImagen) {  
    document.getElementById(id).src=rutaImagen;  
}
```

9. VARIABLES LOCALES Y GLOBALES

Actualmente en javascript existen 3 tipos de variables:

- **var** → Variables en ES5 → Scope de función
- **let** → Incorporado en ES6 → Scope de bloque
- **const** → Incorporado en ES6 → Scope de bloque(*constante*)



VAR → **scope de función**

```
function explainVar(){  
    var a = 10;  
    console.log(a);           // output 10  
    if(true){  
        var a = 20;  
        console.log(a);       // output 20  
    }  
    console.log(a);           // output 20  
}
```

```
var i = 60;  
(function explainVar(){  
    for( var i = 0; i < 5; i++){  
        console.log(i)         //output 0, 1, 2, 3, 4  
    }  
})();  
  
console.log("Despues del loop", i) // output 60
```




LET → scope de bloque

```
function explainLet(){  
  let a = 10;  
  console.log(a);           // output 10  
  if(true){  
    let a = 20;  
    console.log(a);         // output 20  
  }  
  console.log(a);           // output 10  
}
```

CONST → constante con scope de bloque

```
function explainConst(){  
  const x = 10;  
  console.log(x);           // output 10  
  x = 20;                   // throws type error  
  console.log(x);  
}
```

RESUMEN

	<ul style="list-style-type: none">- Es una señal de que una variable no debe ser re-assignada- Nos va a forzar a escribir código más limpio- Scope: bloque. Uso cercano de la variable
	<ul style="list-style-type: none">- Lo usamos para decir que una variable puede ser re-assignada- Generalmente usada para iterar ciclos for o cuando no nos queda otra opción- Scope: bloque. Uso cercano de la variable.
	<ul style="list-style-type: none">- No nos dice mucho. No indica cómo deberían usarse las variables.- Puede presentar una constante o una variable que se puede re-assignar- Puede servir para usar variables dentro de un bloque o dentro de una función, lo cual es confuso y puede traer bugs.