# Tema 01

# JavaScript

# Introducción

## Introducción

Es un lenguaje basado en objetos y guiado por eventos diseñado específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de internet.

El programa que interpreta el lenguaje script es el propio navegador, por tanto, dicho navegador debe de soportar el lenguaje correspondiente de lo contrario no funcionará nada.

La etiqueta que vamos a utilizar para insertar el código de *javascript* es **<script>...</script>.**

Los lenguajes script son lenguajes interpretados. Esto significa que el código se va interpretando línea a línea, es decir, cada línea ha sido traducida a lenguaje máquina en el instante previo a su ejecución y después de su ejecución no se guarda el resultado de la compilación. Por tanto, cada vez que se ejecute el código hay que volverlo a traducir a lenguaje máquina.

Las ventajas e inconvenientes del lenguaje interpretado son las siguientes:

***ventajas***

- *El código es cómodo para depurar, ya que no es necesario volver a compilar tras un cambio.*
- *No es necesario disponer de un compilador, ya que el intérprete ejecuta el script.*
- *El mantenimiento es fácil y rápido, por parte del autor o de otro programador.*

***inconvenientes***

- *La ejecución se ralentiza, al ser necesario la interpretación línea a línea cada vez.*
- *El código es visible y puede ser objeto de plagio por parte de otras personas.*
- *El usuario tiene acceso al código y puede modificarlo, estropeando alguna operación.*

## Mi primer código de ejecución en cliente

Como hemos comentado anteriormente para poder ejecutar código *javascript* hay que introducirlo entre las etiquetas **<script>...</script>** y hay que utilizar el atributo *language* con el valor *JavaScript (En html5 no es necesario)*.

La etiqueta *script* se puede utilizar en cualquier parte del documento. Aunque se aconseja utilizarla entre las etiquetas <*head*> y <*/head*> . También se utilizará entre las etiquetas <*body*> *y* <*/body*> cuando sea necesario.

Ver página http://rolandocaldas.com/html5/como-incluir-javascript-en-html5 para ver donde tenemos que incluir el código javascript (o ver fichero js_Tema01_anexoI.odt)

Un ejemplo de código script sería:

```
<script language="javascript">
{
        alert("hola");
}
</script>
```

## Generalidades

- Es "Case Sensitive"
- Comentarios en "/* ..... */" para bloques y "//" para una línea
- Cada sentencia debe termina en ; (punto y coma) (aconsejable, NO obligatorio)
- El código se agrupa entre llaves "{ ..... }"
- La ejecución es lineal. En el punto donde encuentra un error detiene la ejecución del código.

## Tipos de variables

Es un lenguaje débilmente tipado. Una misma variable puede contener números y luego letras.

El nombre de las variables:

- No debe tener espacios en blanco
- Puede estar formado por números, letras y el carácter subrayado
- No se pueden utilizar palabras reservadas
- El primer carácter debe ser una letra o el símbolo de subrayado _

Para definir una variable se utiliza la palabra *var* delante de la variable o variables a definir.

*var _var1, var2;*

Se puede definir una variable e inicializarla a la vez.

*var var1="cadena inicial";*

## Funciones de javascript

En este apartado veremos la primeras funciones de javasscript. Tiene más funciones que iremos estudiaremos durante el curso.

**alert("texto").** Muestra un mensaje por pantalla.

```
<!DOCTYPE html>
<html>
  <head>
        <title>alert</title>
        <meta charset="UTF-8" />
        <script>
              alert("el primer alert. No ves nada en la página");
        </script>
  </head>
  <body>
        Antes del script en body.
        <script>
              alert("que tal");
        </script>
        Después del script en body.
  </body>
</html>
```

**prompt("texto","defecto").** Función que utilizamos para realizar una petición de datos por pantalla y asignar el valor a una variable. En esta función aparecen dos botones "*Aceptar*" y "*Cancelar*". Si pulsamos *Aceptar,* la variable toma el valor introducido en la caja de texto. Si pulsamos *Cancelar* asigna *null* a la variable.

```
<!DOCTYPE html>
<html>
  <head>
        <title>prompt</title>
         <meta charset="UTF-8" />
  </head>
  <body>
        Ejemplo de prompt
        <script>
              var pulsado;
              pulsado=prompt("Estás de acuerdo?");
              alert(pulsado);
        </script>
  </body>
</html>
```

**confirm("texto").** Se utiliza para realizar una pregunta al usuario y aparecen dos botones *Aceptar* y *Cancelar*. El resultado de esta función se asigna a una variable. Esta variable toma el valor *true* si el usuario pulsa sobre el botón *Aceptar* y asigna *false* si el usuario pulsa sobre el botón *Cancelar*.

```
<!DOCTYPE html>
<html>
  <head>
        <title>confirm</title>
        <meta charset="UTF-8" />
  </head>
  <body>
        Ejemplo de prompt
        <script>
              var pulsado;
              pulsado=confirm("Dime un nombre");
              alert(pulsado);
        </script>
  </body>
</html>
```

**isNaN(valor).** Evalúa un valor y devuelve *verdadero* si el valor evaluado no es número y *false* si el valor evaluado es número. (NaN – Not-a-Number – No es un número).
*valor* puede ser una variable o un valor.

**isFinite(valor).** Devuelve *verdadero* cuando "valor" es un número finito. Cuando es NaN, infinito positivo o infinito negativo devuelve *false*. Si "valor" es un *string* también devuelve *false*.

```
<!DOCTYPE html>
<html>
  <head>
        <title>Ambito variables</title>
        <meta charset="UTF-8" />
  </head>
  <body>
        <script>
              alert(isNaN(15)); // false
              alert(isNaN("15")); // false
              alert(isNaN("a15")); // true
              alert(isNaN("15a")); // true
              alert(isFinite(15)); // true
              alert(isFinite("15")); // true
              alert(isFinite("a15")); // false
              alert(isFinite("15a")); // false
        </script>
  </body>
</html>
```

# Constantes

Experimental ECMAScript 6.

Se declaran con la palabra reservada *const*. Una constante no se puede cambiar de valor una vez declarada y mantiene el valor mientras el código javascript está en ejecución.

El ámbito de una constante es igual que para una variable. Hay que tener en cuenta que si omitimos la palabra reservada *const* se define una variable y no una constante.

```
<!DOCTYPE html>
<html>
<head>
<title>Constantes</title>
<meta charset="UTF-8" />
<script>
	const mensaje1="texto de mensaje global";
	function visualiza()
	{
		const mensaje="texto de mensaje local";
		alert(mensaje); // visualiza mensaje local
		alert(mensaje1); // visualiza mensaje global
	}
	visualiza(); // llama función y visualiza los dos mensajes
	alert(mensaje1); // visualiza mensaje global
	alert(mensaje); // No visualiza nada.
</script>
</head>
<body>
</body>
</html>
```

Si intentamos cambiar el valor de una constante no tiene efecto y el sistema no produce ningún error.

```
<!DOCTYPE html>
<html>
<head>
	<title>Constantes</title>
	<meta charset="UTF-8" />
</head>
<body>
	<script>
		const mensaje="mensaje"
		alert(mensaje);
		const mensaje="ppp";
		alert(mensaje);
	</script>
</body>
</html>
```

No podemos utiliza la palabra var y cont en la misma instrucción

var cont variable=”valor”; // Esto es erróneo.

# *Tipos de datos*

## Enteros

Los número enteros los podemos representar en base 10, en base 16 y en base 8.

Los número en base 8 van precedidos del símbolo **0** (cero)

*01234* es el número $1234_{(8}$

```
alert(0123);

83
```

Los número en base 16 van precedidos del símbolo **0x** (cero equis)

*0xA37B* es el número $A37B_{(16}$

```
alert(0x123);

291
```

## Flotante

Representamos números en coma flotante, usando notación decimal o científica.

*Muestran el mismo número*

*alert(123.456);*

*alert(1.23456E2);*

*Muestran el mismo número*

*alert(0.00123);*

*alert(1.23E-3);*

## Booleanos

Los valores booleanos son *true* (1) o *false* (0).

## Nulos

El valor especial para indicar que el **objeto** no ha sido asignado es *null*

## Indefinidos

El valor *undefined* indica que la **variable** no le ha sido asignado ningún valor, y por tanto permanece indefinida. La variable ha sido declarada pero no se le ha asignado valor.

## Cadenas

Las cadenas pueden ir delimitadas entre comillas simples o comillas dobles.

Cuando se utilicen comillas dobles hay que tener cuidado cuando se utilizan incrustadas en directivas HTML.

Esta línea se ejecuta mal

*<input type="button" onclick="alert("cadena a mostrar")">*

Esta línea se ejecuta bien

*<input type="button" onclick="alert('cadena a mostrar')">*

También podemos utilizar los siguientes caracteres de escape:

| | |
|---|---|
| *\b* | *Espacio hacia atrás* |
| *\f* | *Alimentación de línea* |
| *\n* | *Nueva línea* |
| *\r* | *Retorno de carro* |
| *\t* | *Tabulación* |
| ** | *Barra invertida: * |
| *\'* | *Comilla simple: '* |
| *\"* | *Comilla doble: "* |

# *Operadores*

## Asignación

El operador de asignación es el símbolo = (igual)

Si se escribe *a* = *b* estamos asignando a la variable *a* el valor de la variable *b*.

## Concatenación

El operador de concatenación es el símbolo +

*cadena="cadena derecha " + "cadena izquierda"*

es lo mismo que

*cadena="cadena derecha cadena izquierda"*

## Aritméticos

Tenemos que diferenciar entre operadores unarios y operadores binarios. Un operador unario es cuando solo tiene un único operando. Un operador binario es cuando necesita dos valores.

### *Operadores binarios*

| | |
|---|---|
| + | suma: a + b |
| - | resta: a – b |
| * | producto: a * b |
| / | división: a / b |
| % | Módulo (Resto): a % b |

También existen unos operadores abreviados

| | |
|---|---|
| += | b += 2 equivale a b = b + 2 |
| -= | b -= 2 equivale a b = b - 2 |
| *= | b *= 2 equivale a b = b * 2 |
| /= | b /= 2 equivale a b = b / 2 |
| %= | b %= 2 equivale a b = b % 2 |

### *Operadores unarios*

| | |
|---|---|
| ++ | *variable*++ *ó* ++*variable* |
| -- | *variable*-- *ó* --*variable* |

```
variable1=20;
variable2=variable1++;
```

```
variable1=20;
variable3=++variable1;
```

*variable2* tendrá el valor 20 y *variable3* tendrá el valor 21.

## Comparación

Los operadores de comparación son binarios y devuelven un valor booleano.

| | |
|---|---|
| == | igual |
| != | distinto |
| > | mayor que |
| < | menor que |
| >= | mayor o igual que |
| <= | menor o igual que |
| === | estrictamente igual |
| !== | estrictamente diferente |

En el operador *estrictamente igual* deben de coincidir el tipo y el valor.

```
variable1=20;
variable2=”20”;
alert(variable1 == variable2); // true
alert(variable1 === variable2); // false
```

## Lógicos

Son los comparadores **y, o** y **no**. Se utilizan para hacer condiciones compuestas.

| | |
|---|---|
| && | AND ( 'y' lógico) |
| \|\| | OR ('o' lógico) |
| ! | NOT ('no' lógico) |

| Operador | Descripción |
|---|---|
| . [ ] ( ) | Acceso a campos, índice de matrices y llamada a funciones. |
| ++ -- - ! delete new typeof void | Incremento +1, decremento -1, negativo, NOT lógico borrado, crear objeto, mostrar tipo, indefinido |
| * / % | Multiplicación, división, módulo de división (resto) |
| + - + | Suma, resta, concatenación de cadenas |
| << >> >>> | Bit shifting |
| < <= > >= | menor que, menor que o igual, mayor que, mayor que o igual |
| == != === !== | Igualdad, desigualdad, identidad, no identidad |
| & | AND |
| ^ | XOR |
| \| | OR |
| && | AND logico |
| \|\| | OR logico |
| ?: | Condicional |
| = | Asignación |
| , | Evaluación múltiple |

## Conversión de tipos

**Conversión implícita.** Las variables son débilmente tipadas, por tanto, con asignarle un tipo diferente de dato ya se cambia el tipo de dato.

```
<script>
var cadena="esto es un texto";
alert(typeof(cadena)); // visualiza String
var cadena=15;
alert(typeof(cadena)); // visualiza number
</script>
```

Cuando una cadena está formada exclusivamente por números, al realizar una operación aritmética modifica el tipo de dato.

```
<script>
var cadena="15";
var numero="10";
var result;
result=cadena+numero;
alert("result=cadena+numero --> result es del tipo " + typeof(result));
result=cadena-numero;
alert("result=cadena-numero --> result es del tipo " + typeof(result));
result=cadena*numero;
alert("result=cadena*numero --> result es del tipo " + typeof(result));
result=cadena/numero;
```

```
alert("result=cadena/numero --> result es del tipo " + typeof(result));
result=cadena % numero;
alert("result=cadena%numero --> result es del tipo " + typeof(result));
cadena*= 1;
alert("cadena=cadena * 1 --> cadena es del tipo " + typeof(cadena));
</script>
```

Hay que tener cuidado con el operador +. Este operador también se utiliza para realizar la concatenación de cadenas, por tanto, con este operador no se realiza la conversión.

Si nos fijamos en el último ejemplo. Al multiplicar una variable de tipo string por uno se convierte en una variable de tipo number.

Si tenemos una variable numérica y la concatenamos con un string esta variable se convierte al tipo string.

```
<script>
var cadena="15";
var numero=10;
result=cadena+numero;
alert("result=cadena+numero --> result es del tipo " + typeof(result));
alert("numero es del tipo " + typeof(numero));
numero += "";
alert("numero=numero+" --> numero es del tipo " + typeof(result));
</script>
```

**Conversión explícita.** Para realizar este tipo de conversión utilizaremos dos funciones y un método.

La funciones son parseInt() y parseFloat().

parseInt(), parseFloat(). Convierte una cadena en un número. Si la cadena tiene caracteres no numéricos serán ignorados y solo convertirá los dígitos que hay a la izquierda del carácter no numérico. El espacio en blanco al principio de la cadena no se tiene en cuenta para realizar la conversión

parseInt(cadena,base). Convierte la cadena en números enteros decimales. Los decimales son ignorados.

```
<script>
var texto="23.62";
alert(parseInt(texto)); // visualiza 23
texto="25 años";
alert(parseInt(texto)); // visualiza 25
texto= "a23";
alert(parseInt(texto)); // visualiza NaN
texto= " 32";
alert(parseInt(texto)); // visualiza 32
texto= " 52 57 63";
alert(parseInt(texto)); // visualiza 52
</script>
```

Esta función dispone de un parámetro que podemos utilizar para indicar en que base está representado el número.

```
<script>
var cadena="111";
alert(parseInt(cadena,2)); // visualiza 7
alert(parseInt(cadena,8)); //visualiza 73
alert(parseInt(cadena,16)); //visualiza 273
</script>
```

parseFloat(cadena). Esta función convierte también número fraccionados y sólo dispone de un único parámetro.

**Método .toString(base).** Es un método de objeto Number de javascript. Convierte un número a una cadena.

```
<script>
var numero=25;
alert(numero.toString(2)); // visualiza 11001
alert(numero.toString(8)); // visualiza 31
alert(numero.toString(16)); // visualiza 19
cadena=numero.toString();
alert("tipo de dato " + typeof(cadena) + " valor " + cadena); // visualiza tipo de dato string valor 25
</script>
```

## *Salida de datos*

Vamos a utilizar un método y unas funciones que están indicadas para introducir y mostrar información. Con este método y funciones podremos comunicarnos con el usuario que esté interactuando con la página web.

Lo primero que veremos es como mostrar información al usuario. Para ello utilizaremos un método del objeto *document* (objeto del navegador) que iremos estudiando poco a poco a lo largo de este curso. Bién, el método en cuestión es *write()*, que nos permite escribir en el documento.

*document.write("esto es un mensaje al usuario");*

Aquello que ponemos entre las comillas puede ser texto o etiquetas html, que serán interpretadas correctamente.

*document.write("esto es un ejemplo<br>");*

# Estructuras de control

## Condicionales

### *if*

Esta estructura es *Si --- entonces --- si no ---* . Tiene la siguiente sintaxis:

```
if (condicion) {
        código }
else {
        código
}
```

```
var uno=2;
if(uno == 1)
  alert(“es uno”);
else
  alert(“no es uno)”;
```

También existe una combinación ternaria:

*[expresión] ? [sentencia1] : [sentencia2];*

```
var uno=2;
uno == 1 ? alert("es uno"):alert("no es uno");
```

### *switch*

Estructura de control condicional múltiple. La sintaxis de esta estructura es la siguiente:

```
switch(variable) {
        case caso1:
                sentencias;
                break;
        case caso2:
                sentencias;
                break;
        default:
                sentencias;
                break;

}
```

```
var uno=3;
switch(uno) {
   case 1:
      alert("es uno");
      alert("algo mas");
      break;
   case 2:
      alert("es dos");
      alert("algo mas");
      break;
   default:
      alert("es cualquier cosa");
      alert("algo mas");
      break;
}
```

La variable “uno” debe coincidir en tipo y valor con los valores indicados en las opciones *case.*

El comando *break* en cada opción es para que al llegar a ese punto no continúe ejecutando el resto de instrucciones. Si en *case 1* no ponemos el *break* y la variable *uno* vale *1* ejecutaría todas las instrucciones de *case 1* y de *case 2* sin evaluar la segunda condición.

Al coincidir con una de las opciones ejecutará las instrucciones hasta encontrar un *break*. Después abandona la instrucción, NO sigue buscando coincidencias.

## Bucles

Los bucles se pueden interrumpir con los comando *break* o *continue*. Con *break* se interrumpe el bucle y lo abandona. Con *continue* se dejan de ejecutar las instrucciones de la presente iteración, pero no abandona el bucle.

### *for*

Este bucle se ejecuta mientras cumpla la condición. La sintaxis es la siguiente:

```
for(inicialización;condición;incremento){
        sentencias;
}
```

El orden de ejecución del bucle *for* es el siguiente:

1. Ejecuta la inicialización
2. Revisa la condición
3. Ejecución bloque instrucciones
4. Ejecuta el incremento
5. Vuelve al punto 2

Ninguno de los argumentos es obligatorio, se podría escribir

*for(;;);*

Teniendo en cuenta que esto es un bucle sin fin.

Aquí tenemos un ejemplo del bulce *for*

```
var var1;
for(var1=1;var1<=10;var1++)
{
document.write(var1 + "<br />");
}
```

En el argumento se pueden incluir más de una variable.

```
var var1,var2;
for(var1=1,var2=0;var1<=10 && var2 < 5;var1++, var2++)
{
document.write(var1 + " -- " + var2 + "<br />");
}
```

### *for...in*

Este bucle recorre todas las propiedades de un objeto dado. La sintaxis de esta estructura es la siguiente:

```
for(var variable in objeto) {
   sentencias;
}
```

```
for(var i in navigator){
document.write("navigator." + i +  " = " + navigator[i] + " <br
/>");
}
```

La variable *i* coge, una a una, todas las propiedades del objeto *navigator*.

### *while*

El bucle *mientras* que ya se conoce de programación. Este bucle se puede ejecutar 0 o más veces.

```
while (condición){
   sentencias;
}
```

### *do ...while*

```
do {
   sentencias
} while (condición);
```

El bucle se ejecuta 1 o más veces hasta que se cumpla la condición.