

# UT2. Estructura del lenguaje JavaScript

## 1) Fundamentos de JavaScript.

- 1) Comentarios en el código.
- 2) Variables.
- 3) Tipos de datos
- 4) Operadores.
- 5) Condiciones y bucles.

# Fundamentos de JavaScript I

**JavaScript añade una capa de comportamiento** en el desarrollo de nuestras páginas web.

Hasta JavaScript había una capa para la estructura y otra para la presentación.

- **JavaScript:** Es un lenguaje de programación que te permite controlar el comportamiento de tu página web.
- **CSS:** es el lenguaje utilizado para formatear cada uno de los elementos o grupos de elementos.
- **HTML:** es el lenguaje que define la estructura para dar un sentido al contenido web.

# Fundamentos de JavaScript II

El homólogo de **Microsoft** de JavaScript es **Jscript**.

Le puso otro nombre para **evitar problemas con la marca de JavaScript**, pero tenía muchos **problemas de compatibilidad** con múltiples navegadores.

# Fundamentos de JavaScript IV

**El último standard de JavaScript es ES7 (ECMAScript 2016), pero no está implantado en todos los navegadores, la versión ES6 (ECMAScript 2015) sí, por lo que se recomienda programar cumpliendo este estandard ES6.**

**JavaScript se diseñó con una sintaxis similar al lenguaje C y aunque adopta nombres y convenciones del lenguaje Java.**

**JavaScript fue desarrollado originariamente por Brendan Eich, de Netscape, con el nombre de Mocha, el cual se renombró posteriormente a LiveScript y quedó finalmente como JavaScript.**

**Hoy en día JavaScript es una marca registrada de Oracle Corporation.**

# Fundamentos de JavaScript.

## Comentarios en el código I

**Se aconseja el código autocomentado:**

- Agrupar instrucciones en **funciones con un nombre descriptivo**.
- Utilizar **nombres de variables descriptivas**.
- **Sangrado de código**.

Si aún así, **el código no se entendiera bien se recomienda introducir comentarios**.

Los comentarios **son sentencias que el intérprete de JavaScript ignora**.

**Inconveniente:** los comentarios se transfieren desde el servidor al cliente junto al código, por lo que hace que **la transferencia sea más pesada**.

# Fundamentos de JavaScript.

## Comentarios en el código II

Modos de introducir comentarios:

### 1) Comentar una línea completa:

```
// Esto es un comentario
```

### 2) Comentar parte de una línea:

```
var x=3;    // x guardará el número de elementos
```

### 3) Comentar varias líneas:

```
/* function myFunction(p1, p2) {  
    return p1 * p2;  
} */
```

# Fundamentos de JavaScript.

## Comentarios en el código III

Utilidades de los comentarios:

- 1) Desactivar un bloque de código.**
- 2) Introducir versión, fecha del código, autoría, etc.**

# Fundamentos de JavaScript.

## Variables I

Una variable es **un espacio de memoria** en el que se almacena información.

En JavaScript **no se indica el tipo de variable**. El tipo lo adquiere al asignarle un valor. Cambia de tipo al asignarle un valor de otro tipo.

**Forma de declarar variables:**

- `var x,y,z;    // Se declaran varias variables a la vez`
- `var num;        // Observa que no se indica el tipo`  
    `// Variable sin inicializar.`
- `var num=2;                //Se declara y se inicializa. El tipo es number.`
- `var nombre="Ana";    //El tipo será string.`

**No es necesario declarar una variable** para utilizarla, **pero no es una buena práctica.**

```
num=3;    //en "strict mode" no funcionaría
```



# Fundamentos de JavaScript.

## Variables II

**Las variables se nombran con:**

- Caracteres alfanuméricos.
- Guión bajo.

**No se pueden utilizar:**

- Signos de puntuación.
- Espacios en blanco.
- Palabras reservadas por el lenguaje como return, if, function, ....

**Técnica de nombrado de variables LowerCamelCase:**

```
var nombrePersonaContacto;
```

# Fundamentos de JavaScript.

## Tipos de datos I

**Existen los siguientes tipos de datos:**

- **string:** una serie de caracteres entre doble comillas o comillas simples
- **number:** cualquier tipo de número (entero, coma flotante, ...)
- **boolean:** true o false.
- **object:** dato con propiedades y métodos.
- **function:** las funciones es un tipo de dato.
- **undefined:** una variable que está vacía tiene como tipo de datos undefined.

**Para averiguar el tipo de datos: `typeof variable;`**

# Fundamentos de JavaScript.

## Tipos de datos I

**Existen los siguientes tipos de object:**

- **string:** cuando el string se crea así:

```
var cadena=new String("hola")
```

- **number:** cuando el number se crea así:

```
var num=new Number(3)
```

- **boolean:** cuando el boolean se crea así:

```
var b=new Boolean(true)
```

- **Date:** para guardar fechas. Se crea así:

```
var fecha=new Date()    // guarda la fecha actual
```

- **Array:** se puede crear de dos formas:

```
var arr=["1","2"];      var=new Array("1","2");
```

- **null:** objeto vacío.

# Fundamentos de JavaScript.

## Tipos de datos II. string

```
var nombre="Ana", apellido='Perez';
```

**Se utiliza \ para escapar caracteres especiales**

```
var cad="\ ' Juan\ ' "
```

```
var cad="1\n2\n3" //Salto de línea
```

```
var cad="1\t2\t3" //Tabulador
```

**Propiedad de la longitud de la cadena:**

```
cad.length
```

**Concatenar cadenas:**

```
cad="hola" + "clase";
```

# Fundamentos de JavaScript.

## Tipos de datos III. number

```
var num1 = 3.49876;    //Coma flotante
var num2 = 16;          //Entero
var num3 = 2.8e4;       //Notación científica
var num4 = 4.5e-2;      //Notación científica
var num5 = 0234;        // base 8 (octal)
var num6 = 0x2A34;      // base 16(hexadecimal)
```

# Fundamentos de JavaScript.

## Tipos de datos IV. boolean


```
var resultado = true //Booleano a true
```

```
var resultado = false //Booleano a false
```

# Fundamentos de JavaScript.

## Tipos de datos V. object



```
var persona = {  
    nombre: "Marta",  
    apellido: "Perez",  
    edad: 42  
}  
  
//persona.nombre sería "Marta"  
//persona.edad sería 42
```

# Fundamentos de JavaScript.

## Tipos de datos VI. object. array

Un array es un tipo de objeto:

```
var ciudad = ["Madrid", "Alcobendas", "Sevilla"];  
// ciudad[0] sería "Madrid"  
// ciudad[2] sería "Sevilla"
```



# Fundamentos de JavaScript.

## Tipos de datos VII. function



```
function suma (a,b) {  
    return a+b;  
  
}  
  
var x;  
  
x=suma(3,4);
```

# Fundamentos de JavaScript.

## Conversión de datos I. De número a string.

```
var x = 43;
```

```
x.toString(); //resultado es "43"
```

```
 var y = 4 + "3" // resultado es "43"
```

```
var y = 4 + 2 + "3" // resultado es "63"
```

```
var y = 4 + 2 + "3" + 3 + 9 // resultado-> "6339"
```



# Fundamentos de JavaScript.

## Conversión de datos II. De string a número.

```
parseInt ("10"); //devuelve nº entero 10  
parseFloat ("10.33") //da nº flotante 10.33  
parseInt ("dos"); //devuelve NaN
```

**NaN (Not a Number) es un valor de tipo number que representa "Not a Number" **

Es devuelto por funciones que no pueden operar con los parámetros introducidos en la función.



# 1.4. Operadores (I)

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

[http://www.htmlpoint.com/javascript/corso/js\\_30.htm](http://www.htmlpoint.com/javascript/corso/js_30.htm)



# 1.4. Operadores (II).

## Operadores de comparación

Operadores de comparación en JavaScript			
Sintaxis	Nombre	Tipos de operandos	Resultados
==	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 que.	Todos.	Boolean.
<=	Menor o igual que.	Todos.	Boolean.

<http://www.webestilo.com/javascript/js06.phtml>

# 1.4. Operadores (III).

## Operadores de comparación. Ejemplos

### Comparar números

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

### Comparar cadenas

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



En este caso JS convierte cada carácter a su valor ASCII

### Comparar números con su cadena

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

```
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.

# 1.4. Operadores (IV).

## Operadores aritméticos

Operadores aritméticos en JavaScript			
Sintaxis	Nombre	Tipos de Operando	Resultados
+	Más.	integer, float, string.	integer, float, string.
-	Menos.	integer, float.	integer, float.
*	Multiplicación.	integer, float.	integer, float.
/	División.	integer, float.	integer, float.
%	Módulo.	integer, float.	integer, float.
++	Incremento.	integer, float.	integer, float.
--	Decremento.	integer, float.	integer, float.
+valor	Positivo.	integer, float, string.	integer, float.
-valor	Negativo.	integer, float, string.	integer, float.

# 1.4. Operadores (V).

## Operadores aritméticos. 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.
```

```
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.
```



# 1.4. Operadores (VI).

## Operadores de asignación

Operadores de asignación en JavaScript			
Sintaxis	Nombre	Ejemplo	Significado
=	Asignación.	x = y	x = y
+=	Sumar un valor.	x += y	x = x + y
-=	Substraer un valor.	x -= y	x = x - y
*=	Multiplicar un valor.	x *= y	x = x * y
/=	Dividir un valor.	x /= y	x = x / y
%=	Módulo de un 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 derecha 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 asignaciones.	[a,b]=[c,d]	a=c, b=d

# 1.4. Operadores (VII).

## Operadores booleanos

Debido a que parte de la programación tiene un gran componente de lógica, es por ello, que los operadores booleanos juegan un gran papel. Los operadores booleanos te van a permitir evaluar expresiones, devolviendo como resultado (verdadero) o false (falso).

Operadores de boolean en JavaScript			
Sintaxis	Nombre	Operandos	Resultados
<b>&amp;&amp;</b>	AND.	Boolean.	Boolean.
<b>  </b>	OR.	Boolean.	Boolean.
<b>!</b>	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
```

# 1.4. Operadores (VIII).

## Operadores booleanos

Tabla de valores de verdad del operador AND			
Operando Izquierdo	Operador AND	Operando Derecho	Resultado
True	&&	True	True
True	&&	False	False
False	&&	True	False
False	&&	False	False

Tabla de valores de verdad del operador OR			
Operando Izquierdo	Operador OR	Operando Derecho	Resultado
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
```



# 1.4. Operadores (IX).

## Operadores bit a bit

-Para los programadores de scripts, las operaciones bit a bit suelen ser un tema avanzado.

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.

# 1.4. Operadores (X).

## Operadores bit a bit

Tabla de operador Bit a Bit en JavaScript			
Opera dor	Nombre	Operando izquierdo	Operando derecho
&	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.

# 1.4. Operadores (XI).

## Operadores bit a bit

En el siguiente enlace podrás ver ejemplos de operaciones a nivel de bits:

[http://www.mundoprogramacion.com/net/dotnet/operar\\_con\\_bits.aspx](http://www.mundoprogramacion.com/net/dotnet/operar_con_bits.aspx)

# 1.4. Operadores (XII).

## Operadores de objeto

Este grupo de operadores se relaciona directamente con **objetos y tipos de datos**. La mayor parte de ellos fueron implementados a partir de las **primeras versiones**

*. (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.método()`.

Ejemplo con un objeto nativo de JavaScript:

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

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

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";`

# 1.4. Operadores (XIII).

## Operadores de objeto

*Delete (para eliminar un elemento de una colección).*

Por ejemplo si consideramos:

```
var oceanos = new Array("Atlantico", "Pacifico", "Indico","Artico");
```

Podríamos hacer:

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

Ésto 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 a la izquierda del operador, es una cadena referente a la propiedad o método (simplemente el nombre del método sin paréntesis); 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`



# 1.4. Operadores (XIV).

## Operadores de objeto

*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 validando.  
}
```

# 1.4. Operadores (XV).

## 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`.

# 1.4. Operadores (XVI).

## Operadores misceláneos

### *? : (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);  
}
```

# 1.5. Condiciones y bucles (I)

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.

Se puede ejecutar unas u otras instrucciones, dependiendo de ciertas condiciones, y cómo puedes repetir una o varias instrucciones, las veces que te hagan falta.

## *Estructuras de control.*

*Construcción if*

*Construcción if ... else*

## *Bucles.*

*Bucle for.*

*Bucle while().*

*Bucle do ... while()*



# 1.5. Condiciones y bucles (II)

## Estructuras de control

- Permiten **controlar las decisiones** y bucles de ejecución.
- **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** .

*Construcción if*

*Construcción if ... else*

# 1.5. Condiciones y bucles (III)

## Estructuras de control

### 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`.

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.");
}
```

# 1.5. Condiciones y bucles (IV)

## Estructuras de control. switch

**Equivale a varias construcciones `if...else` anidadas, pero bastante más legible.**

```
switch (expresión) { // expresión a evaluar
  case valor1:
  case valor2:
    // instrucciones a ejecutar si la evaluación de
    // la expresión coincide con alguno de los valores
  break;
    // importante para romper el flujo
  case valorN:
  case valorN+1:
    // instrucciones a ejecutar si la evaluación de la
    // expresión coincide con valorN, valorN+1
  break: // importante para romper el flujo

  default: // instrucciones a ejecutar si la evaluación de
           // la expresión no coincide con algún valor.
}
```

# 1.5. Condiciones y bucles (V)

## Estructuras de control. switch

**Ejemplo:**

Ejemplo:

```
switch (tipoIVA) {  
    case 4:  
        alert("Superreducido");  
        break; // si no está este break, si tipoIVA es 4,  
               // se ejecutará también alert("Reducido")  
               // hasta encontrarse con un break.  
    case 10: alert("Reducido");  
            break;  
    case 21: alert("General");  
            break;  
    default: alert("Valor de IVA incorrecto");  
}
```



# 1.5. Condiciones y bucles (VI)

## 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.*

Este tipo de bucle te deja repetir un bloque de instrucciones un número limitado de veces.

Sintaxis:

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

Ejemplo:

```
for (var i=1; i<=20; i++)
{
    // instrucciones que se ejecutarán 20 veces.
}
```

# 1.5. Condiciones y bucles (VII)

## Bucles

### *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, como veremos a continuación, 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 i=0;
while (i <=10)
{
    // Instrucciones a ejecutar dentro del bucle hasta que i sea mayor que 10 y no se cumpla
    la condición.
    i++;
}
```

# 1.5. Condiciones y bucles (VIII)

## Bucles

### *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()` visto anteriormente. 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);
```

# 1.5. Condiciones y bucles (IX)

## Ejemplo sencillo

**Ejercicio :** Prueba el siguiente código (lo tienes en las páginas 14-15 de los apuntes) . Abre el editor web y crea un archivo **.html** .

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo Básico de JavaScript</title>

</head>
<body>
<h2>Código fuente de un ejemplo básico con JavaScript
</h2>
<script type="text/javascript">
// Definimos variables.
var nombre, apellidos, edad, hermanos;

nombre="Rafa";
apellidos="Martinez Diaz";
edad=38;
hermanos=3;

// Imprimo el nombre y los apellidos.
document.write("Hola " + nombre + " " + apellidos);

// Imprimo un salto de línea.
document.write("<br/>");

// Imprime la edad y el número de hermanos.
document.write(nombre + " tienes " + edad + " años y además tienes " + hermanos + "
hermanos.<br/>");

// Fijate en la diferencia entre las dos siguientes líneas.
document.write("Dentro de 15 años tu edad será " + edad + 15 + "<br/>");
document.write("Dentro de 15 años tu edad será " + (edad+15) + "<br/>");

// Tu nombre escrito 50 veces.
for (i=1; i<=50; i++)
{
    document.write(nombre + ",");
}
</script>
</body>
</html>
```

# Ejemplos w3schools

**Ejercicio** : Consulta la siguiente información y ejemplos de JavaScript del tema en la página web de w3schools:

<http://www.w3schools.com/js/default.asp>

En concreto, debes consultar los apartados siguientes

***-Data Types***

***-Variables***

***-Output***

***-Operators***

***-Arithmetic***

***-Assignment***