

Unidad 2.- Manejo de la sintaxis del lenguaje:

- a) Constantes y variables.
- b) Tipos de datos.
- c) Asignaciones.
- d) Operadores.
- e) Comentarios al código.
- f) Sentencias.
- g) Decisiones.
- h) Bucles.

JavaScript es un lenguaje de programación que fue desarrollado en sus inicios para el navegador web Netscape Navigator y creado por Brendan Eich en 1995.

JavaScript fue diseñado para que su código se incluyese en documentos HTML y fuera interpretado por el navegador web. Ha cumplido con tanto éxito su propósito que el consorcio responsable de la especificación del lenguaje HTML (W3C ⇒ World Wide Web Consortium <http://www.w3c.org> en inglés y <http://www.w3c.es> en español), entre otras cosas, lo ha elegido como estándar para HTML5. Actualmente la responsable del desarrollo de JavaScript es la Mozilla Foundation <http://www.mozilla.org/foundation/>, con la documentación del lenguaje en inglés en <https://developer.mozilla.org/en-US/docs/JavaScript> y en castellano en <https://developer.mozilla.org/es/docs/JavaScript>.

Además existe una estandarización de JavaScript realizada por el ECMA (European Computer Manufacturers Association) Internationala través del ECMAScript 262, que se puede consultar en la página web <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.

Deberemos tener en cuenta ciertas cuestiones a la hora de realizar programas en este lenguaje de programación.

- Una cosa muy importante a tener en cuenta cuando se programa en JavaScript es que los distintos navegadores van a poder tener diferencias en el lenguaje de programación ya que realizan diferentes interpretaciones del estándar.
- Diferencia las letras mayúsculas de las letras minúsculas, de tal forma que los siguientes nombres en un programa en JavaScript representan distintos elementos: MARIA, Maria, maria, MariA,...
- Todas o casi todas las instrucciones de JavaScript terminan en un punto y coma.
- Una instrucción puede ocupar una o varias líneas.
- En una línea se pueden incluir varias instrucciones, siempre y cuando estén separadas por un punto y coma.
- Varios espacios consecutivos son considerados como si fuese un único espacio.

a) Constantes y variables.

Una constante es un valor que no se puede modificar, permanece inalterable a lo largo de la ejecución del programa.

Para declarar constantes vamos a utilizar

const nombre-constante-1=valor-1 [, nombre-constante-2=valor-2] ...

001	<code>const</code> apellidos = "Muñoz Bayón";
002	<code>const</code> nombre = "Félix Ángel";

Las constantes van a ser del tipo de datos correspondiente al valor que se la asigna.

Una variable es una zona de la memoria donde vamos a alojar un dato y al cual vamos a poder hacer referencia a través de un nombre. El dato que vamos a tener almacenado puede variar a lo largo de la ejecución del programa.

JavaScript es un lenguaje débilmente tipado, esto nos indica que una variable va a ser del tipo de datos correspondiente al dato que tiene almacenado. Es más, a una variable le vamos a poder asignar valores de diferentes tipos de datos.

Para declarar una variable vamos a poner
var *nombre-variable*;

001	var nombre;
-----	--------------------

o bien

let *nombre-variable*;

La diferencia entre declarar una variable con var o con let va a ser el ámbito u alcance de la misma. Mientras que con let el ámbito es el bloque de código en el que se declara, con var su ámbito es la función dentro de la que se encuentra.

001	let modulo;
-----	--------------------

Al mismo tiempo que la variable se declara se la puede asignar un valor a través de

var *nombre-variable* = valor ;
let *nombre-variable* = valor ;

001	var apellido="Muñoz";
-----	------------------------------

002	let modulo="cliente";
-----	------------------------------

En una misma declaración vamos a poder declarar varias variables para lo cual usaremos

var *nombre-variable-1* , *nombre-variable-2* , ... ;
let *nombre-variable-1* , *nombre-variable-2* , ... ;

001	var edad, domicilio;
-----	-----------------------------

al mismo tiempo que se declaran varias variables se pueden inicializar

var *nombre-variable-1* [= valor-1],[*nombre-variable-2* [= valor-2]]... ;
let *nombre-variable-1* [= valor-1],[*nombre-variable-2* [= valor-2]]... ;

001	var edad=35 , domicilio="mi casa";
-----	---

b) Tipos de datos.

En este lenguaje tenemos diferentes tipos de datos que son:

- ◆ Numéricos, **Number** que almacena números enteros y reales en notación decimal (136), hexadecimal (0x136) o científica (13.6e8). Los valores que pueden tomar este tipo de datos van comprendidos entre **Number.MIN_VALUE** y **Number.MAX_VALUE**, además de estos valores, también pertenecen a este tipo de datos los valores **Infinity**, **-Infinity** y **NaN** (Not a Number). **Number(valor)** devuelve el valor numérico del valor que representa.

- ◆ Cadenas, **String** es una cadena de caracteres encerrada entre comillas o apostrofes. Los limitadores deben ser iguales, el inicial y el final, no se pueden mezclar. Incluso pueden ocupar varias líneas, basta con poner al final de cada línea el carácter '\n'.
- ◆ Plantillas de cadenas es una cadena que va delimitada por el carácter del acento grave "`", puede ocupar más de una fila y no hace falta poner ningún carácter al final de las líneas.
- ◆ Lógicos o Booleanos solamente admite los valores **true** o bien **false**.
- ◆ Undefined este tipo de datos solamente admite el valor **undefined**, valor que se asigna de forma automática a todas aquellas variables que se han declarado y que no se la ha asignado valor.
- ◆ **Object**, representa un objeto genérico.
- ◆ Null que solamente puede tomar el valor **null**, éste valor se suele asignar a los objetos.

La clase **Number** posee las siguientes propiedades:

- ◆ **MAX_VALUE**: valor máximo.
- ◆ **MIN_VALUE**: valor mínimo
- ◆ **NaN**: no es un número
- ◆ **NEGATIVE_INFINITY**: infinito negativo, para overflow.
- ◆ **POSITIVE_INFINITY**: infinito positivo, para overflow.

Un objeto Number también dispone de los siguientes métodos:

- ◆ **toExponential([número-decimales])**: devuelve una cadena con el número en notación científica, exponencial. Se pueden indicar el número de decimales de la mantisa, si es menor de los que tiene realiza un redondeo.
- ◆ **toFixed([número-decimales])**: devuelve una cadena con el número en notación de coma fija inicialmente sin decimales, se puede indicar cuantos decimales se desean mostrar.
- ◆ **toString()**: devuelve una cadena con el valor del número.

001	<code>var y=Number("36");</code>
002	<code>var z= 78;</code>

Un objeto o una variables **String** posee la siguiente propiedad

- ◆ **length**: longitud de la cadena.
- ◆ **prototype**: nos sirve para definir a continuación más propiedades o métodos.

Un objeto o una variables **STRING** posee los siguientes métodos.

- ◆ **charAt(posición)**: devuelve el carácter de la cadena que ocupa la posición indicada.
- ◆ **charCodeAt(posición)**: devuelve el número que representa el carácter que ocupa la posición indicada en unicode .
- ◆ **concat(lista-cadenas)**: devuelve una cadena que es la unión de la cadena y la lista de cadenas.

- ◆ **indexOf(cadena [, posición]):** devuelve la posición en que se encuentra la primera aparición de la cadena indicada, devuelve -1 si no la encuentra. La búsqueda se inicia en el principio a no ser que se indique la posición a partir de la cual se quiere realizar la búsqueda.
- ◆ **lastIndexOf(cadena [, posición]):** devuelve la posición en que se encuentra la primera aparición de la cadena indicada, devuelve -1 si no la encuentra. La búsqueda se inicia en el final a no ser que se indique la posición a partir de la cual se quiere realizar la búsqueda. La búsqueda se realiza de final a principio.
- ◆ **includes(cadena [, posición]):** devuelve un valor lógico que nos indica si la cadena se encuentra dentro de la variable. La búsqueda se inicia en el principio a no ser que se indique la posición a partir de la cual se quiere realizar la búsqueda.
- ◆ **search(cadena):** devuelve la posición que ocupa en la cadena inicial la cadena indicada.
- ◆ **substr(posición, número-caracteres):** devuelve una cadena con tantos caracteres como se indican a partir de la posición indicada.
- ◆ **substring(posición-1 [, posición-2]):** devuelve una cadena con los caracteres existentes entre las posiciones indicadas, excepto el carácter que ocupa la posición indicada por posición-2.
- ◆ **slice(posición-1 [, posición-2]):** copia de la cadena inicial todos los caracteres que hay desde la posición-1 a la posición-2 excluido; si se omite posición-2 es hasta el final y si posición-2 es negativo se indican el número de caracteres del final que no se copian.
- ◆ **toLowerCase():** devuelve una copia de la cadena con las letras en minúsculas.
- ◆ **toUpperCase():** devuelve una copia de la cadena con las letras en mayúsculas.
- ◆ **toString([base]):** devuelve una cadena con el valor del objeto, si el valor es numérico se indica la base, por defecto 10.
- ◆ **fromCharCode(lista-códigos-unicode):** devuelve una cadena con los caracteres indicados a través de sus código unicode.
- ◆ **replace(cadena-reemplazar, cadena-nueva):** devuelve una nueva cadena en la cual a la cadena inicial se la ha sustituido la cadena-reemplazar por la cadena-nueva.
- ◆ **trim():** devuelve la cadena sin los blancos de la izquierda ni de la derecha.
- ◆ **valueOf():** valor de la cadena.
- ◆ **padEnd(longitud):** devuelve la cadena con al menos la longitud indicada, poniendo espacios al final de la cadena, si es necesario.
- ◆ **padStart(longitud):** devuelve la cadena con al menos la longitud indicada, poniendo espacios al inicio de la cadena, si es necesario.

Métodos de la clase **String** para HTML

- ◆ **anchor(nombre):** devuelve una cadena con el resultado de crear un enlace, etiqueta <a> con el literal de la cadena inicial y cuyo atributo name es el nombre indicado.
- ◆ **big():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <big>.
- ◆ **blink():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <blink>. opera y firefox.
- ◆ **bold():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta .
- ◆ **fixed():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <tt>.
- ◆ **fontcolor(color):** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta .
- ◆ **fontsize(tamaño):** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta .
- ◆ **italics():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <i>.
- ◆ **link(url):** devuelve una cadena con el resultado de crear un enlace, etiqueta <a> con el literal de la cadena inicial y cuyo atributo href es la url indicada.
- ◆ **small():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <small>.
- ◆ **strike():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <strike>.
- ◆ **sub():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <sub>.
- ◆ **sup():** devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <sup>.

Una variable de tipo string puede ser manejada como un array.

<code>var uno=new String("dato"); //crea objeto String</code>
<code>var dos=String("primero"); // crea una cadena</code>
<code>var tres="prueba"; //crea una cadena</code>

c) Asignaciones.

Las asignaciones nos permiten introducir un valor en una variable, este valor puede venir de una expresión.

Operadores de asignación

=	Asignación	Asigna el valor de la expresión a la variable
+=	Suma/concatenación y asignación	Suma el valor de la expresión al valor de la variable y luego asigna el valor a la misma. A+=3. Incrementa el valor de A en 3.
-=	Resta y asignación	Resta el valor de la expresión al valor de la

		variable y luego asigna el valor a la misma. $A = 3 * B$. equivale a $A = A - (3 * B)$
*=	Multiplicación y asignación	Multiplica el valor de la expresión con el valor de la variable y luego asigna el valor a la misma.
/=	División y asignación	Divide el valor de la variable entre el valor de la expresión y luego asigna el cociente a la variable.
%=	Módulo y asignación	Divide el valor de la variable entre el valor de la expresión y luego asigna el resto a la variable.
**=	Potencia y asignación	Realiza la potencia del valor de la variable elevado al valor de la expresión

d) Operadores.

Los operadores nos permiten realizar ciertas operaciones con los operandos, vamos a tener varios tipos de operadores, que son:

Operadores aritméticos:

++	Incremento	incrementan en uno al operando, puede ir delante del operando (preincremento, primero se incrementa el operando y luego se realiza la expresión donde se encuentra ++a) o bien detrás del operando (postincremento, primero se realiza la expresión donde se encuentra y luego se incrementa el operando a++). $A = 5;$ $B = ++A * 3; // B = 6 * 3 = 18, A=6.$ $C = A++ * 5; // C = 6 * 5 = 30. A=7.$
--	Decremento	Decrementa en uno al operando, puede ir delante del operando (predecremento, primero se decrementa el operando y luego se realiza la expresión donde se encuentra --a) o bien detrás del operando (postdecremento, primero se realiza la expresión donde se encuentra y luego se decrementa el operando a--). $A = 5;$ $B = --A * 3; // B = 4 * 3 = 12. A=4.$ $C = A-- * 5; // C = 4 * 5 = 20. A=3.$
+	Suma	suma los operando situados a su lado.
-	Resta	resta al operando de izquierda el valor del operando de su derecha.
*	Multiplicación o producto	multiplica los dos operandos
/	División o cociente división.	divide el operando de su izquierda entre el operando de su derecha.
%	Resto división	realiza la división entera y devuelve el resto.

	entera o módulo	
**	Potencia	

Operadores booleanos o lógicos.

!	Negación	niega el operando de su derecha.
&&	Y lógico o AND	solo devuelve verdadero si los dos son verdaderos y falso en caso contrario.
	O lógico u OR.	devuelve verdadero si uno de sus operandos es verdadero y falso en caso contrario.

Operadores de bits

&&	Y lógico o AND	
	O lógico u OR.	
^	XOR	
~	Complementación o negación	
<<	Desplazamiento a la izquierda	Desplaza los bits a la izquierda. 16 << 2. Al número 16 le desplaza dos bits a la izquierda.
>>	Desplazamiento a la derecha	Desplaza los bits a la derecha insertando cero por la izquierda. 24 >> 3. Al número 24 le desplaza tres bits a la derecha.
>>>	Desplazamiento a la derecha, insertando ceros por la izquierda	Si el número es positivo es igual al anterior, si es negativo no.

Operadores relacionales

==	Igual que	!!!Cuidado al comparar expresiones de números reales!!! 0.3 == 0.2 + 0.1 es falso
!=	Distinto que	
===	Idéntico, incluso en tipo	
!==	Diferente a (tipo y/o valor)	
<	Menor que	
<=	Menor o igual que	
>	Mayor que	
>=	Mayor o igual que	

Operador de cadenas

+	Concatenación	
---	---------------	--

Otros operadores

Condición?valor1:valor2	Operador	Si la condición es verdadera
-------------------------	----------	------------------------------

	condicional	devuelve el valor1 y si es falsa devuelve el valor2.
typeof <i>expresión</i>	Indica el tipo de la expresión	

e) Comentarios al código.

Existen dos formas de poner comentarios en Javascript, que son:

Comentarios de una línea, para lo cual utilizaremos:

// *comentario*

todo lo que va detrás de las dos barras y hasta el final de línea es un comentario.

Comentarios de más de una línea.

/*

comentario

***/**

todo lo que hay entre los caracteres de inicio (**/***) y los de final (***/**) es un comentario, puede ocupar una o varias líneas.

f) Sentencias.

Vamos a ver algunas sentencias que vamos a ir utilizando como son:

```
alert( cadena );  
window.alert( cadena );
```

Nos muestra un cuadro de mensaje con la cadena indicada.

```
confirm( cadena );  
window.confirm( cadena );
```

Nos muestra un cuadro de mensaje con la cadena y con los botones de Aceptar o Cancelar, devuelve true o 1 para Aceptar y false o 0 para Cancelar.

```
prompt( cadena );  
window.prompt( cadena );
```

Nos muestra un cuadro de mensaje con la cadena y nos pide un valor, nos muestra también los botones de Aceptar o Cancelar. Devuelve un valor string con el valor introducido si se pulsa Aceptar y con el valor nulo, null, si se pulsa el botón de cancelar.

```
console.log(cadena);  
window.console.log(cadena);
```

Nos muestra la cadena indicada en la consola del navegador web.

document.write(cadena);

Escribe en el documento el contenido de la cadena, si la cadena incluye código HTML, las etiquetas son interpretadas por el navegador.

La instrucción para detección de errores es:

```
try {
    instrucciones-1 }
[ catch ( variable-excepción if condición ) { ...
    instrucciones-2 } ]
[ catch ( variable-excepción ) { ...
    instrucciones-3 } ]
[ finally {
    instrucciones-n } ]
```

Si al ejecutar las instrucciones1 se produce un error o una excepción se ejecutan las instrucciones que acompañan al **catch** correspondiente a ese error o excepción, sino se encuentra se ejecutan las instrucciones-3. Además siempre, que se incluyan, al final se van a ejecutar las instrucciones que hay a continuación de **finally**.

Para generar excepciones tenemos la instrucción **throw** cuya forma es:

throw excepción ;

Genera una excepción representada por el valor de la expresión, que puede ser una cadena, un número o un valor lógico.

g) Decisiones.

Las instrucciones de decisión nos permiten ejecutar unas instrucciones en función de que se cumpla una determinada condición. Para ello tenemos la instrucción **if** con sus diferentes formatos.

```
if ( condición ) {
    instrucciones-1
} else {
    instrucciones-2
}
```

si la condición es verdadera, se ejecutan las instrucciones-1, mientras que si la condición es falsa se ejecutan las instrucciones-2, siempre y cuando se incluyan las mismas.

001	<code>cadena=window.prompt("Introduce una palabra");</code>
002	<code>if (cadena=="felix") {</code>

003	window.alert('Nombre Correcto');
004	} else {
005	window.alert("Nombre erroneo");
006	}

Siguiente formato

```

if ( condición-1) {
    instrucciones-1
} else if ( condición-2) {
    instrucciones-2
} [ else if ( condición-3) {
    instrucciones-3
} ] ...

[ else {
    instrucciones-n
} ]

```

Se ejecutarán las instrucciones existentes a continuación de la primera condición que sea verdadera, si todas las condiciones son falsas se ejecutarán las instrucciones existentes a continuación del último **else**.

001	cadena=window.prompt("Introduce una letra");
002	if (cadena=="a") {
003	window.console.log('Primera vocal "a"');
004	} else if (cadena=="e") {
005	window.console.log("Segunda vocal 'e'");
006	} else if (cadena=="i") {
007	window.console.log("Tercera vocal 'i'");
008	} else if (cadena=="o") {
009	window.console.log("Cuarta vocal 'o'");
010	} else if (cadena=="u") {
011	window.console.log("Quinta vocal 'u'");
012	} else {
013	window.console.log("Es una consonante");
014	}

La alternativa múltiple la tenemos a través de sentencia switch cuyo formato es:

```

switch ( expresión ) {
    case valor-1:
        instrucciones-1
        break;
    [ case valor-2:
        instrucciones-2 ] ...
        break;
    [ default:
        instrucciones-n
    ]
}

```

Se van a ejecutar las instrucciones existentes a continuación del primer valor que se encuentra y que coincide con el valor de la expresión y dejará de ejecutar las instrucciones cuando encuentre la instrucción **break**; Si ninguno de los valores que se muestran coincide con el valor de la expresión se ejecutan las instrucciones existentes a continuación de **default**, si se incluye.

001	<code>cadena=window.prompt("Introduce un dígito");</code>
002	<code>switch (cadena) {</code>
003	<code> case "1" :</code>
004	<code> case "3" :</code>
005	<code> case "5" :</code>
006	<code> case "7" :</code>
007	<code> case "9" :</code>
008	<code> window.console.log("dígito impar");</code>
009	<code> break;</code>
010	<code> case "0" :</code>
011	<code> window.console.log("dígito cero");</code>
012	<code> break;</code>
013	<code> case "2" :</code>
014	<code> case "4" :</code>
015	<code> case "6" :</code>
016	<code> case "8" :</code>
017	<code> window.console.log("dígito par");</code>
018	<code> break;</code>
019	<code> default :</code>
020	<code> window.console.log("No es un dígito");</code>
021	<code>}</code>

h) Bucles.

Los bucles nos van a permitir repetir un conjunto de instrucciones un número determinado de veces.

La primera instrucción que nos permite realizar bucles es la instrucción **while** cuyo formato es:

```
while ( condición ) {
    instrucciones
}
```

Si la condición es verdadera se van a ejecutar las instrucciones, una vez ejecutadas se vuelve a evaluar la condición y mientras que la condición sea verdadera se van a ejecutar las instrucciones. Si la primera vez que se evalúa la condición es falsa, las instrucciones nunca se llegarán a ejecutar.

001	<code>var numero=1;</code>
002	<code>window.console.log("Primer bucle");</code>
003	<code>while (numero > 10) {</code>
004	<code> numero++;</code>
005	<code>}</code>
006	<code>window.console.log(numero);</code>
007	<code>window.console.log("Segundo bucle");</code>
008	<code>while (numero < 10) {</code>
009	<code> numero++;</code>
010	<code>}</code>
011	<code>window.console.log(numero);</code>

Otra instrucción que tenemos para bucles es la sentencia **do..while**, cuyo formato es.

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

Las instrucciones se van a ejecutar y luego se va a evaluar la condición, si la misma es verdadera, las instrucciones se seguirán ejecutando mientras que la condición sea verdadera. En este caso las instrucciones se ejecutarán al menos una vez.

001	window.console.log("Tercer bucle");
002	var segundo=1;
003	do {
004	segundo++;
005	} while(segundo > 10);
006	window.console.log(segundo);
007	window.console.log("Cuarto Bucle");
008	do {
009	segundo++;
010	} while (segundo < 10);
011	window.console.log(segundo);

La última instrucción que tenemos para la realización de iteraciones es la instrucción **for**, cuyo formato es:

```
for( inicialización; condición-fin ; incremento ) {  
    instrucciones  
}
```

En primer lugar se realiza la inicialización, a continuación si la condición es verdadera se ejecutan las instrucciones, a continuación se realiza la incrementación y si la condición es verdadera se vuelve a ejecutar las instrucciones y así repetidamente mientras que la condición sea verdadera.

001	window.console.log("Quinto Bucle");
002	for (var indice=1 ; indice < 10; indice++) {
003	window.console.log(indice);
004	}
005	window.console.log("Sexto Bucle");
006	for (var indice=15 ; indice > 0; indice--) {
007	window.console.log(indice);
008	}
009	window.console.log("Septimo Bucle");
010	for (var indice=0 ; indice < 41; indice+=2) {
011	window.console.log(indice);
012	}
013	window.console.log("Octavo Bucle");
014	for (var indice=37 ; indice > 1; indice-=3) {
015	window.console.log(indice);
016	}

Dentro de los bucle tenemos dos instrucciones que son:

break;

Nos permite salir en ese instante del bucle dentro del cual se encuentra.

continue;

Salta a comprobar la condición del bucle para determinar si se va a seguir ejecutando o bien va a finalizar.

Estas dos instrucciones no se deben utilizar a no ser casos extremos, ya que deseamos aplicar técnicas de programación estructurada.

Para objeto tenemos los siguientes **for**, que veremos más detenidamente en el tema de objetos.

```
for( variable in objeto ) {  
    instrucciones  
}
```

```
for( variable of objeto ) {  
    instrucciones  
}
```