Sintaxis del lenguaje JavaScript

1. Introducción

JavaScript fue desarrollado originalmente por la empresa Netscape, tomando como referencia los lenguajes Java y C.

Diferencias entre JavaScript y Java:

JavaScript	Java
Herencia basada en prototipos	Herencia basada en clases
Lenguaje no tipado (los tipos de datos se infieren en tiempo de ejecución)	Lenguaje fuertemente tipado
No puede realizar operaciones de lectura/escritura en discos locales	Puede realizar operaciones de lectura/escritura en discos locales

En 1997 se produce la primera versión del estándar (**ECMAScript**) por ECMA Internacional.

W3C (World Wide Web Consortium) publicó un modelo de objetos **DOM** (Document Object Model) para garantizar la compatibilidad entre navegadores a la hora de manipular el contenido de una página web.

2. Comentarios en el código

Podemos usar comentarios a nivel de línea y de múltiples líneas.

- **Línea**: usando la doble barra //

  var nombre=”Mensaje de pruebas”; // Declaración de un variable

- **Multilínea**: usando /* y */

  /* Comentario multilínea cuyo texto es
  ignorado completamente por el intérprete
  JavaScript */

3. Sintaxis del lenguaje JavaScript

3.1. Lenguaje sensitivo a mayúsculas y minúsculas (case sensitive)

JavaScript distingue entre mayúsculas y minúsculas. Por tanto, la variable dato será completamente distinta a la variable Dato.

El siguiente código produciría un error:

var dato=5;

alert(“El valor de dato es “ + Dato);

Lo mismo ocurre con los identificadores de función. En el siguiente código la función mensaje sería distinta de la función Mensaje:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>JavaScript</title>
    <meta charset="utf-8" >
    <script>
   function mensaje(){
      alert("Hola!");
    }
    function Mensaje(){
     alert("Adiós!");
    }
</script>
</head>
<body>

<p>JavaScript Template</p>

<input type="button" onclick="mensaje();" value="Pulsar" />
<input type="button" onclick="Mensaje();" value="Pulsar" />


</body>
</html>
```

(Código en fichero ej_2.4_1.html)

3.2. Fin de instrucción

Una instrucción puede extenderse varias líneas, y para señalar el fin de instrucción se suele utilizar el símbolo “;”, aunque es opcional (será obligatorio si declaramos varias instrucciones en una sóla línea de código).

El siguiente ejemplo ilustra los conceptos anteriores:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>JavaScript</title>
    <meta charset="utf-8" >
    <script>
    /* Delimitación de sentencias en JavaScript
    Utilización del ; y de los saltos de línea
    */
```

```
10.    function mensaje(){
11.       precio="300€";cantidad=3; //Uso de ; para separar sentencias
12.       mensaje="datos" // No usamos el delimitador ;
13.       alert('precio:' + precio + ' cantidad:' + cantidad
14.       + ' y mensaje:' + mensaje)
15.      }
16. </script>
17. </head>
18. <body>
19.
20. <p>JavaScript Template</p>
21. <h2> Diferencia entre mayúsculas y minúsculas </h2>
22. Ejemplo delimitación sentencias:<input type="button" onclick="mensaje();" value="Pu
    lsar" />
23.
24. </body>
25. </html>
```

(Código en fichero **ej_2.4_2.html**)

3.3. No tiene en cuenta los espacios en blanco y las nuevas líneas

El intérprete de JavaScript ignora cualquier espacio en blanco sobrante.

De hecho, las librerías como JQuery tienen dos versiones, la versión uncompressed, legible para los humanos y la versión compressed, de menor tamaño sin espacios ni saltos de línea:

A continuación se muestra código JavaScript en formato legible, adecuado para ser entendido y mantenido por un programador:

Versión uncompressed: http://code.jquery.com/jquery.js

El siguiente código es equivalente al anterior pero se han quitado los espacios, saltos de línea, tabulación, y se han renombrado las variables para obtener un código más compacto. Este código se llama ofuscado y no es apto para ser mantenido por un programador:

Versión compressed: http://code.jquery.com/jquery.min.js

3.3.0.1. [G.M. 23/9/2024]

4. Declaración de variables

En JavaScript la declaración de variables tiene dos características importantes:

- **La declaración de variables es opcional (pero es muy recomendable declararlas)**
- **JavaScript es un lenguaje débilmente tipado:** al declarar una variable no se especifica el tipo como en Java, sino que se infiere al cargar un determinado tipo de datos, además el tipo de dato cargado en una variable puede variar a lo largo de la ejecución del programa.

4.1. Tipos de declaraciones de variables

Existen dos tipos de declaraciones:

- **Declaración explícita :** declaramos expresamente la variable
- **Declaración implícita:** no declaramos la variable, sino que nos limitamos a asignarle un valor.

4.1.1. Declaración explícita

En este caso declaramos la variable usando la palabra clave var o let:

```
var identificador = valor;
```

Dependiendo del lugar donde se realice la declaración de la variable tendremos variables de ámbito local (visible sólo dentro de una función) o global.

En la versión de ES6 se introdujo let, que además de soportar el ámbito local y global también soporta el de bloque (por ejemplo dentro un while, if, for, etc.):

```
let identificador = valor;
```

Es más recomendable usar let frente a var, ya que la regla general en programación es declarar el ámbito de una variable con el menor alcance posible.

4.1.2. Declaración implícita (no se debe usar)

En la declaración implícita realizamos una asignación a una variable directamente, sin usar var o let:

```
identificador = valor;
```

El alcance de esta nueva variable será definido global (este es precisamente el motivo por el que no debe usarse)

4.2. Declaración de constantes

La declaración de constantes involucra la palabra reservada const y el dato almacenado en la constante no podrá modificarse a lo largo de la ejecución del programa:

```
const PI = 3.1416;
```

4.3. Reglas para los identificadores de variables

Las especificaciones de JS impone las siguientes restricciones para los identificadores de variables:

- Solamente pueden estar formados por letras, números, el símbolo $ (dollar) y el símbolo _ (guión bajo).

- El primer carácter no puede ser nunca un número. El identificador puede comenzar por letra, por el símbolo $ o por el símbolo _.

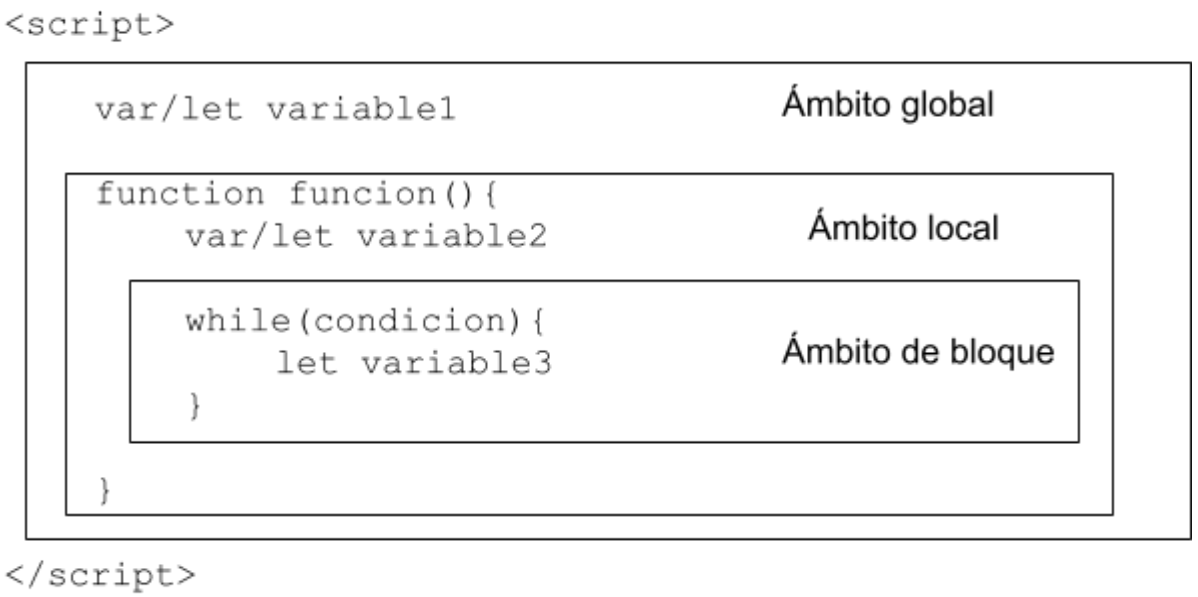
Ejemplos:

```
let $edad=21; // Correcto
let _nombre=”Javier”; // Correcto
let apellido=”Soldado”; // Correcto
let 2019factura = “Ventas”; // Incorrecto, no puede comenzar por
número
let mi@email = “profesorjaviersoldado@hotmail.com”; //Incorrecto,
el identificador de variable no puede contener el símbolo @
```

4.4. Ámbito de ejecución de una variable

Cuando una variable es declarada de manera explícita existirán tres ámbitos de ejecución dependiendo de en qué zona del código la hayamos declarado:

- **Global:** podrá ser utilizada en cualquier parte del código. La declaración se realizará fuera de toda función. También son globales las variables declaradas implícitamente (sin usar var).
- **Local:** las variables declaradas localmente solamente se pueden utilizar dentro de la función donde son declaradas.
- **Bloque:** las variables definidas dentro de un bloque (while, foreach, for, while, etc.) sólo podrán ser accedidas desde dentro de dicho bloque. Para que realmente el ámbito sea de bloque habrá que usar **let** para declarar la variable (si se usa **var** el ámbito será a nivel de función, no de bloque)



4.5. Tipos de datos

JavaScript maneja los siguientes tipos de datos:

- Numérico (**number**) ⇒ `let edad= 30; let iva=0.21;`
- Booleano (**boolean**): podrán contener true o false ⇒ `let fumador = true;`
- Cadena de caracteres (**string**) ⇒ `let nombre = "Javier";`
- Objeto (**object**) ⇒ `let persona = { nombre: "Javier", apellido: "Soldado", edad=46 };`
- Nulo (**null**): una variable a la que se le asigna el valor null será realmente de tipo object ⇒ `let obj = null;`
- No definido (**undefined**): variable declarada pero a la que no se ha asignado ningún valor ⇒ `let altura;`

Identificación de tipos (typeof)

Para conocer el tipo de dato que contiene una variable podemos usar la función typeof(variable). Esta función devolverá el tipo: number, string, object, boolean, etc.

```
1.  <!DOCTYPE html>
2.  <html lang="es">
3.  <head>
4.      <title>JavaScript</title>
5.      <meta charset="utf-8" >
6.      <script>
7.          var edad=21; //Variable con valor numérico entero
8.          var euro=166.386; // Variable con valor numérico decimal
9.          var dato; //Variable con valor undefined
10.         var obj=null; //Variable con valor null
11.         var fumador=true; //Variable con valor booleano
12.         var nombre="Javier"; //Variable con valor string
13.         var persona={         //Variable de tipo objeto
14.             nombre:"Javier",
15.             apellido:"Soldado"
16.             };
17.         function vertipos(){
18.             alert("typeof edad=" + typeof edad + "\n" +
19.                   "typeof euro=" + typeof euro + "\n" +
20.                   "dato=" + typeof dato + "\n" +
21.                   "typeof obj=" + typeof obj + "\n" +
22.                   "typeof nombre=" + typeof nombre + "\n" +
23.                   "typeof persona=" + typeof persona + "\n");
24.         }
25.     </script>
26. </head>
27. <body>
28.
29. <p>JavaScript Template</p>
30. <h2> Ver tipos de datos </h2>
31. <input type="button" onclick="vertipos();" value="Pulsar" />
32.
33. </body>
34. </html>
```

(Código en fichero ej_2.6.7_1.html)

JavaScript al ser un lenguaje débilmente tipado realiza conversiones automáticamente, pero habrá que ser cauto al utilizarlas para no que se generen errores inesperados:

Conversión automática según lo esperado:

```
var a = “resultado es “;
var b = 25;
var c = a + b;
alert (c); // La variable b es convertida a automáticamente a string y c contendrá el string “resultado es 25”;
```

Conversión automática según lo esperado:

```
var a = “12“;
var b = 3;
var c = a - b;
alert (c); // La variable a es convertida a automáticamente a number y c contendrá el number 9;
```

Conversión automática falla:

```
var a = “hola“;
var b = 3;
var c = a - b;
alert (c); // La variable a no puede ser convertida a number y c contendrá NaN (Not a Number)
```

5. Operadores

5.1. Operadores de asignación

Se utiliza para asignar un valor a una variable. Ej:

```
var nombre = “Javier”;
var fumador = false;
```

5.2. Operadores aritméticos

Los operadores aritméticos básicos son:

- Suma: **+** ⇒ var a= 10; var b=7; var c = a + b;
- Resta: **-** ⇒ var a= 10; var b=7; var c = a - b;
- Producto: ***** ⇒ var a= 10; var b=7; var c = a * b;
- División: **/** ⇒ var a= 10; var b=7; var c = a / b;

Tenemos otros operadores adicionales:

Operador	Sintaxis
++ **Incremento**	x++ Retorna el valor actual de x y después actualiza el incremento sobre x sumando un 1 ++x Actualiza el valor actual x y retorna el nuevo valor actualizado con la suma correspondiente
-- **Decremento**	x-- Retorna el valor actual de x y después actualiza el decremento sobre x restando 1 --x Actualiza el valor actual x y retorna el nuevo valor actualizado con la resta correspondiente
%	% Módulo. La operación 7%2 retornaría el valor 1
+=	x += y
-=	x -= y
*=	x *= y
/=	x /= y

5.3. Operadores de comparación

El resultado de la comparación es siempre un valor booleano, el cual nos indica si la comparación es cierta (true) o no (false):

Operador	Descripción
Iguales (==)	Devuelve el valor booleano true si ambos operadores son iguales
Distintos (!=)	Devuelve el valor booleano true si ambos operadores son distintos
Mayor que (>)	Devuelve el valor booleano true si el operador izquierdo es mayor que el derecho
Mayor o igual que (>=)	Devuelve el valor booleano true si el operador izquierdo es mayor o igual que el derecho
Menor que (<)	Devuelve el valor booleano true si el operador derecho es mayor que el izquierdo
Menor o igual que (<=)	Devuelve el valor booleano true si el operador derecho es mayor o igual que el izquierdo

Si en la comparación deseamos que se tenga en cuenta no sólo los valores sino también los tipos de operandos usaremos los siguientes operadores de comparación estricta:

Operador	Descripción
Strict equal (===)	Devuelve el valor booleano true si el operador derecho es igual que el izquierdo y además coinciden los tipos
Strict not equal (!==)	Devuelve el valor booleano true si el operador derecho es distinto que el izquierdo o bien no coinciden los tipos

```
1.  <!DOCTYPE html>
2.  <html lang="es">
3.  <head>
4.      <title>JavaScript</title>
5.      <meta charset="utf-8" >
6.      <script>
7.          function comparadores(){
8.          var edad = 21;
9.          var edadstr="21";
10.         alert(edad == edadstr); // Comparador no estricto => Devuelve true
11.         alert( edad === edadstr); // Comparador estricto => Devuelve false, porque
    los operandos son de distinto tipo (string y number)
12.         }
13.
14.
15.     </script>
16. </head>
17. <body>
18.
19. <p>JavaScript Template</p>
20. <h2> Comparadores </h2>
21. <input type="button" onclick="comparadores();" value="Pulsar" />
22.
23. </body>
24. </html>
```

(Código en fichero **ej_2.7.3_1.html**)

5.4. Operadores lógicos

Los operadores lógicos son ampliamente utilizados en estructuras de control de tipo condicional o bucles, donde la ejecución depende del resultado de una expresión booleana.

Los principales operadores lógicos son:

- **Operador && (AND):**

  El operador && tiene la siguiente tabla de verdad:

A	B	AND
False	False	**False**
False	True	**False**
True	False	**False**
True	True	**True**

  Ejemplo:
  var edad = 21;
  var titulado = true;
  if (edad >= 18 && titulado) alert (“Contratado!”);

- **Operador || (OR):**

  El operador || tiene la siguiente tabla de verdad:

A	B	OR
False	False	**False**
False	True	**True**
True	False	**True**
True	True	**True**

  Ejemplo:
  var bebido = true;
  var drogado = false;
  if (bebido || drogado) alert (“No puede conducir!”);

- **Operador ! (NOT):**

  El operador ! tiene la siguiente tabla de verdad:

!	True	False
	False	True

Ejemplo:
var fumador = false;
if (!fumador) alert (“Enhorabuena por no fumar!”);

6. Estructuras condicionales if

Están formadas por un if . De manera opcional pueden además contener varios else if y un else.

Ejemplo de uso:

```
let num1, num2;
num1 = parseInt(prompt('Ingrese el primer número:'));
num2 = parseInt(prompt('Ingrese el segundo número:'));
if (num1 > num2) {
    alert('el mayor es ' + num1);
} else if(num1 < num2) {
    alert('el mayor es ' + num2);
}
else {
    alert('Ambos números son iguales');
}
```

7. Estructura switch

La instrucción switch es una alternativa para reemplazar en algunas situaciones los if/else if.

Ejemplo de uso:

```
let valor;
valor = parseInt(prompt('Ingrese un valor comprendido entre 1 y
3:'));
switch (valor) {
    case 1:
        alert('uno');
        break;
    case 2:
```

```
        alert('dos');
        break;
    case 3:
        alert('tres');
        break;
      default:
        alert('debe ingresar un valor comprendido entre 1 y 3.');
}
```

8. Estructura repetitiva for

Esta estructura se emplea en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita.

Ejemplo de uso:

```
// La salida se muestra en consola (pulsar F12)
for (let i = 1; i <= 10; i++) {
    console.log("i="+i);
}
```

9. Estructura repetitiva while

Con la estructura while el bloque de código entre llaves se repite MIENTRAS la condición sea Verdadera.

Ejemplo de uso:

```
// La salida se muestra a la consola
let x;
x = 1;
// El bucle while se repita mientras se cumpla la condición
indicada entre paréntesis
while (x <= 100) {
    console.log("x=" + x);
    x = x + 1;
}
```

10. Estructura repetitiva do/while

La sentencia do/while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while que puede no ejecutar el bloque.

Ejemplo de uso:

```
let valor;
do {
valor = parseInt(prompt('Ingrese un valor entre 0 y 999:',
''));
    if (valor < 10) {
        console.log('El valor ' + valor + ' tiene 1 digitos');
    } else if (valor < 100) {
            console.log('El valor ' + valor + ' tiene 2 digitos');
    } else if (valor < 1000) {
            console.log('El valor ' + valor + ' tiene 3 digitos');
    } else {
        console.log('El valor ' + valor + ' tiene más de 3
digitos');
    }
} while (valor != 0);
```

11. Funciones

Una función es un conjunto de instrucciones que resuelven una parte del problema y que puede ser utilizado (llamado) desde diferentes partes de un programa.

Consta de un nombre y parámetros. Los parámetros son valores que se envían como entrada a la función.

Tienen la siguiente estructura:

```
function <nombre de función>(arg1, arg2, ..., arg n)
{
  <código de la función>
}
```

Los parámetros son valores que se envían como entrada a la función. Nos permiten que el código de la función sea reutilizable y llamar a la misma usando distintos valores.

Para que la función retorne un valor usaremos la palabra clave return.

Ejemplo de uso:

El siguiente código hace uso de una función que calcula el área de un rectángulo a partir de dos argumentos (los dos lados del mismo) y retorna el área del mismo haciendo uso de return.

```
// Función con dos parámetros: l1 y l2
// Retorna el producto de los valores recibidos
function areaRectangulo(l1,l2){
   return l1*l2;
}
let lado1;
let lado2;
let area;
lado1=parseFloat(prompt("Introduzca valor lado1"));
lado2=parseFloat(prompt("Introduzca valor lado2"));
area = areaRectangulo (lado1, lado2); // Invocamos a la función pasándole dos argumentos
alert("El área del rectángulo es " + area);
```