UD 02. SINTÁXIS JAVASCRIPT

Índice de contenido

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 linea
-->
</SCRIPT>
```

En Javascript los comentarios se pueden hacer con // para una linea y con /* */ para varias líneas.

Asimismo desde Javascript es posible re-escribir 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 multilinea */
-->
</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"/>
```

Durante el curso trabajaremos con Javascript en ficheros separados.

2. 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 instrucción 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;
```

2.1 Tipos de variables

Los tipos de valores que puede contener una variable JavaScript son:

- 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”,”lalO”). Para concatenar cadenas puede usarse el operador + (“Soy” + “ el “ + “ AMO” formaría la cadena “Soy el AMO”). Una cadena realmente es un array de caracteres (ver array más abajo).

Ejemplo:

```
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");
```

2.2 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 linea
var otroArray=[1,2,"Cancamusa"];
// Valores dentro del array
alert(miVector[1]);
alert(otroArray[2]);
// Valor completo del array
alert(miVector);
// Tamanyo del array
alert(miVector.length);
```

2.3 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.

Ejemplo

```
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. Si hubiera habido parte decimal la truncaria
var n2=parseFloat(num); // Almacena un decimal
var n3=num3.toString(); // Almacena una cadena
```

3. MOSTRAR POR PANTALLA Y LEER DEL TECLADO

3.1 alert()

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 CPIFP");
```

3.2 confirm()

A través del método confirm() se activa un cuadro de diálogo que contiene los botones Aceptar y Cancelar. Cuando el usuario pulsa el botón Aceptar, este método devuelve el valor true; Cancelar devuelve el valor false. 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 ("¿Desea cancelar la subscripción?");
alert("Usted ha contestado que "+respuesta);
```

3.3 prompt()

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. 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.

Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las variables. Algunos 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.

Ejemplo

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

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*

Ejemplo

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

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
==	Devuelve el valor *true* cuando los dos operandos son iguales.
!=	Devuelve el valor *true* cuando los dos operandos son distintos.

>	Devuelve el valor *true* cuando el operando de la izquierda es mayor que el de la derecha.
<	Devuelve el valor *true* cuando el operando de la derecha es menor que el de la izquierda.
>=	Devuelve el valor *true* cuando el operando de la izquierda es mayor o igual que el de la derecha.
<=	Devuelve el valor *true* cuando el operando de la derecha es menor o igual que el de la izquierda.

Ejemplo

```
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));
```

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.*

Suponemos para los ejemplos la siguiente instrucción *var a=3;b=4;c=5;*

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

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

```
alert(El resultado de la expresión 'false&&false' es igual a "+(false&&false));
alert("El resultado de la expresión 'false&&true' es igual a "+(false&&true));
alert("El resultado de la expresión 'true&&false' es igual a "+(true&&false));
alert("El resultado de la expresión 'true&&true' es igual a "+(true&&true));
alert("El resultado de la expresión 'false||false' es igual a "+(false||false));
alert(“El resultado de la expresión 'false||true' es igual a "+(false||true));
alert("El resultado de la expresión 'true||false' es igual a "+(true||false));
alert("El resultado de la expresión 'true||true' es igual a "+(true||true));
alert("El resultado de la expresión '!false' es igual a "+(!false));
```

5. ESTRUCTURAS DE CONTROL Y BUCLES

5.1 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*** es permite la ejecución de un bloque u otro de instrucciones en función de una condición.

Sintaxis:

```
if (condición) {
bloque de instrucciones que se ejecutan si la condición se cumple
}
else{
bloque de instrucciones que 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.

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***.

Ejemplo

```
var xdia;
xdia=prompt("Introduce el día de la semana ", "");
if (xdia == "domingo")
{
      alert("Hoy es festivo");
}
else
      alert ("Hoy no es domingo, es muy probable que tengas que trabajar");
```

Ejemplo

```
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);
}
```

Ramificaciones anidadas

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

Ejemplo

```
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);
```

5.2 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 del índice; Condición de prueba; Modificación en el índice){
      …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");
```

5.3 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.

Sintaxis:

```
while (condición){
      ...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: Pregunta una clave hasta que se corresponda con una dada.

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

5.4 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.

Sintaxis:

```
do {
...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");
```

5.5 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.

La instrucción break dentro de un bucle hace que este 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;
//Mientras no introduzca la clave y no se pulse Cancelar
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");
```

El efecto que tiene la instrucción **continue** en un bucle es el de hacer 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. Esto permite saltarse recorridos del bucle.

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);
}
```

6. FUNCIONES

Una función es un conjunto de instrucciones que se agrupan bajo un nombre de función. Se ejecuta solo cuando es llamada por su nombre en el código del programa. La llamada provoca la ejecución de las órdenes que contiene.

Las funciones son muy importantes por diversos motivos:

- Ayudan a estructurar los programas para hacerlos su código más comprensible y más fácil de modificar.
- Permiten repetir la ejecución de un conjunto de órdenes todas las veces que sea necesario sin necesidad de escribir de nuevo las instrucciones.

  Una función consta de las siguientes partes básicas:

- Un nombre de función.
- Los parámetros pasados a la función separados por comas y entre paréntesis.
- Las llaves de inicio y final de la función.

Sintaxis de la definición de una función:

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

Sintaxis de la llamada a una función:

La función se ejecuta siempre que se ejecute la sentencia.

```
valorRetornado=nombrefuncion (parám1, parám2...);
```

Es importante entender la diferencia entre definir una función y llamarla:

- Definir una función es simplemente especificar su nombre y definir qué acciones realizará en el momento en que sea invocada, mediante la palabra reservada function.
- Para llamar a una función es necesario especificar su nombre e introducir los parámetros que queremos que utilice. Esta llamada se puede efectuar en una línea de órdenes o bien a la derecha de una sentencia de asignación en el caso de que la función devuelva algún valor debido al uso de la instrucción return.

La definición de una función se puede realizar en cualquier lugar del programa, pero se recomienda hacerlo al principio del código o en un fichero “js” que contenga funciones.

La llamada a una función se realizará cuando sea necesario, es decir, cuando se demande la ejecución de las instrucciones que hay dentro de ella.

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){
      // Esta funcion devuelve un valor
      return a+b;
}
function profe (){ // Esta funcion muestra un texto, pero no devuelve un valor
      alert ("El profesor es: Sergi Garcia");
}
// Codigo que se ejecuta
var op1=5;op2=25;
var resultado;
// Llamada a funcion
resultado=suma(op1,op2); //
llamada a la funcion
console.log (op1+"+"+op2+"="+resultado);
// Llamada a funcion
profe();
```

🗲 Atención: recordad que las funciones rara vez hacen funciones de entrada/salida. El 99.9% de las veces simplemente procesan la entrada por parámetros y devuelven un valor.

7. VARIABLES LOCALES Y GLOBALES

Ahora que ya conocemos las funciones es muy importante diferenciar entre variables globales y locales:

- Variables globales: pueden utilizarse en cualquier parte del código y son declaradas fuera de toda función.
- Variables locales: se definen con la instrucción var dentro de una función y solo pueden ser utilizadas dentro de esta.

Ejemplo:

```
var vbleglobal1=20;
function prueba(){
      var vblelocal1=10; //Definición de variable local
      var vblelocal2=vbleglobal1+vblelocal1;
      //En la función se puede acceder a las variables globales y locales
      //definidas dentro de ella
      alert ("La suma de la variable local (10) y la global (20) es "+
      vblelocal2);
}
// Llamamos a la funcion
prueba();
alert ("La variable global es "+vbleglobal1);
//Desde fuera de la función las variables locales definidas en ella no son
accesibles
```

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. MATERIAL ADICIONAL

[1] Curso de Javascript AdaLovecode https://www.youtube.com/watch?v=O9CKO86teyM&list=PLI7nHlOIIPOJtTDs1HVJABswW-xJcA7_o

[2] Intro to Javascript (Udacity) https://eu.udacity.com/course/intro-to-javascript--ud803