FUNCIONES EN JAVASCRIPT

Una función no es más que un bloque de enunciados que componen un comportamiento que puede ser invocado las veces que sea necesario.

SINTAXIS GENERAL DE UNA FUNCIÓN

Una función de JavaScript presenta este aspecto:

```
function nombre_de_la_función(){
    …enunciados a ejecutar…
}
```

Para ejecutar la función posteriormente no hay más que invocar su nombre en cualquier momento y desde cualquier parte de un código, con una excepción: la función debe haber sido definida anteriormente. Así, por ejemplo, este código:

```
function dame_una_a(){
    alert("¡AAAAAAAAAAAAAAAA!");
}

dame_una_a();
```

Ejecutaría la alerta, pero éste:

```
dame_una_a();

function dame_una_a(){
    alert("¡AAAAAAAAAAAAAAAA!");
}
```

Generaría un error, porque en el momento en que se invoca la función ésta aún no ha sido registrada.

Hay que poner especial atención a la hora de crear la funciones, para no repetir los nombres, principalmente porque esto no genera errores en JavaScript, y puede suponer quebraderos de cabeza cuando un script no funciona pero la consola de errores no muestra mensaje alguno.
Si definimos dos funciones con el mismo nombre, como en este ejemplo:

```
function alerta(){
    alert("Ia Ia Shub-Niggurath");
}

function alerta(){
    alert("Klaatu barada nikto");
}

alerta();
```

Sólo funciona la segunda, que ha sido la última definida.

FUNCIONES Y ARGUMENTOS

Podemos desear que una función ejecute unos enunciados en los que opere con una serie de valores que no hayamos definido dentro de la misma, sino que los reciba de otra función o enunciado. Esos valores son los argumentos, que se especifican entre los paréntesis que van tras el nombre de la función, y se separan por comas:

```
function nombre_de_la_función(argumento1,argumento1,…){
    …enunciados a ejecutar…
}
```

Los argumentos se nombran como las variables:

```
function sumar(x,y){
    var total = x + y;
    alert(total);
}
```

Si después se ejecuta unas líneas como las siguientes:

```
sumar(1,2);
sumar(3,5);
sumar(8,13);
```

en la función sumar total adquiere sucesivamente los valores de 3, 8 y 21, que es lo que mostrarían tres alertas.

Aunque en los ejemplos emplee numerales, como argumentos se puede enviar cualquier variable. Sólo hay que recordar que si se trata de una cadena literal, debe ir entrecomillada:

```
la_función('cadena','otra_cadena');
```

Por último, sobre los argumentos hay que recordar las respuestas a tres preguntas:

- ¿Qué ocurre si se envía a una función menos argumentos que los que se han especificado?: Los argumentos que no han recibido un valor adoptan el de undefined.

- ¿Qué ocurre si se envía a una función más argumentos que los que se han especificado?: Los argumentos que sobran son elgantemente ignorados.

- ¿Cuántos argumentos se pueden especificar como máximo?: 255.

VARIABLES Y SU ÁMBITO

El ámbito sería algo así como el espacio en el que las variables existen y al que pueden acceder enunciados u otras funciones. Si se define una variable dentro de una función, esa variable sólo existe para esa función, y otras funciones no pueden acceder a su valor a menos que lo reciban como un argumento:

```
function concatenar_cadenas(){
    var a = "orda";
    var b = "lía";
    var c = a + b;
}

function mostrar_resultado(){
    alert(c);
}

concatenar_cadenas();
mostrar_resultado();
```

En este caso no obtenemos una alerta con «ordalía», sino un error, puesto que en la función mostrar_resultado() pedimos que se muestre el valor de una variable que no existe en su ámbito. Dicho de otra manera, las variables existen sólo para concatenar_cadenas(), y aunque adquieren los valores definidos en cuanto ejecutamos la función, ésta es una «barrera» que impide que se pueda acceder a aquellos desde fuera.

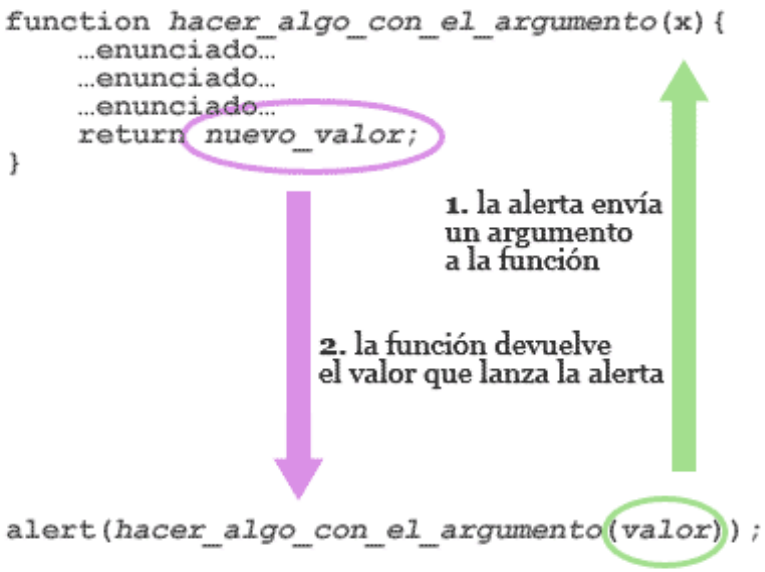
Para que el código funionase, deberíamos definir las variables fuera de ambas funciones, para que su ámbito sea global:

```
var a = "";
var b = "";
var c = "";

function concatenar_cadenas(){
    a = "orda";
    b = "lía";
    c = a + b;
}

function mostrar_resultado(){
    alert(c);
}

concatenar_cadenas();
mostrar_resultado();
```

De esta forma, toda función puede acceder a las variables. concatenar_cadenas() modifica los valores iniciales, pero ahora estos se almacenan de manera global fuera de ella; así están disponibles para mostrar_resultado().

RETURN

Hay otra opción con la que solucionar el problema anterior, y que además puede ser útil en otras ocasiones: return que permite devolver valores al origen de la invocación de una función. El esquema de funcionamiento es algo así:



Un ejemplo esquemático del funcionamiento de return

Rehagamos el ejemplo anterior que no funcionaba por cuestiones de ámbito de las variables, pero enviando el resultado por medio de return:

```
function concatenar_cadenas(){
    var a = "orda";
    var b = "lía";
    var c = a + b;
    return c;
}

function mostrar_resultado(){
    alert(concatenar_cadenas());
}

mostrar_resultado();
```

Ahora mostrar_resultado cuenta con el alert que tiene que mostrar el resultado de la concatenación. En la alerta invocamos la función concatenar_cadenas, que devuelve el valor de c; así, es como si en hubiéramos hecho una alerta de c directamente.