DWEC03.- CONTENIDOS.

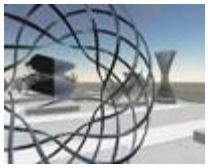
1. Creación de funciones

CASO PRÁCTICO.

Juan está siguiendo los progresos de Antonio, y le recomienda empezar a ver funciones. Las funciones son una herramienta muy potente en los lenguajes de programación, ya que le van a permitir realizar tareas de una manera mucho más organizada, y además le permitirán reutilizar un montón de código en sus aplicaciones.

Antonio verá como crear funciones, cómo pasar parámetros, el ámbito de las variables dentro de las funciones, cómo anidar funciones y las funciones predefinidas en JavaScript. Antonio está muy ilusionado con este tema, ya que a lo largo de las unidades anteriores ha estado utilizando funciones y es ahora el momento en el que realmente verá toda la potencia que le pueden aportar a sus aplicaciones.

En unidades anteriores ya has visto y utilizado alguna vez funciones. **Una función es la definición de un conjunto de acciones pre-programadas**. Las funciones se llaman a través de eventos o bien mediante comandos desde nuestro script.



Siempre que sea posible, tienes que diseñar funciones que puedas reutilizar en otras aplicaciones, de esta forma, tus funciones se convertirán en pequeños bloques constructivos que te permitirán ir más rápido en el desarrollo de nuevos programas.

Si conoces otros lenguajes de programación, quizás te suene el término de *subrutina o procedimiento*. En JavaScript no vamos a distinguir entre **procedimientos** (que ejecutan acciones), o **funciones** (que ejecutan acciones y devuelven valores). **En JavaScript siempre se llamarán funciones**.

Una función es capaz de devolver un valor a la instrucción que la invocó, pero ésto no es un requisito obligatorio en JavaScript. Cuando una función devuelve un valor, la instrucción que llamó a esa función, la tratará como si fuera una expresión.

Te mostraré algunos ejemplos en un momento, pero antes de nada, vamos a ver la sintaxis formal de una función:

```
function nombreFunción ( [parámetro1]... [parámetroN] )
{
        // instrucciones
}
```

Si nuestra función va a **devolver algún valor** emplearemos la palabra reservada **return**, para hacerlo.

Ejemplo:

```
function nombreFunción ( [parámetro1]... [parámetroN] )
{
        // instrucciones
        return valor;
}
```

Los nombres que puedes asignar a una función, tendrán las mismas restricciones que tienen los elementos HTML y las variables en JavaScript. Deberías asignarle un nombre que realmente la identifique, o que indique qué tipo de acción realiza. Puedes usar palabras compuestas como chequearMail o calcularFecha, y fíjate que las funciones suelen llevar un verbo, puesto que las funciones son elementos que realizan acciones.

Una recomendación que te hacemos, es la de que las funciones sean muy específicas, es decir que no realicen tareas adicionales a las inicialmente propuestas en esa función.

Para realizar una llamada a una función lo podemos hacer con:
nombreFuncion( ); // Esta llamada ejecutaría las instrucciones programadas dentro de la función.

Otro ejemplo de uso de una función en una asignación:
variable=nombreFuncion( ); // En este caso la función devolvería un valor que se asigna a la variable.

Las funciones en JavaScript también son objetos, y como tal tienen métodos y propiedades. Un método, aplicable a cualquier función puede ser toString(), el cuál nos devolverá el código fuente de esa función.

2 Parámetros

Cuando se realiza una llamada a una función, muchas veces es necesario pasar parámetros (también conocidos como argumentos). Este mecanismo nos va a permitir enviar datos entre instrucciones.

Para pasar parámetros a una función, tendremos que escribir dichos parámetros entre paréntesis y separados por comas.

A la hora de definir una función que recibe parámetros, lo que haremos es, escribir los nombres de las variables que recibirán esos parámetros entre los paréntesis de la función.

Veamos el siguiente ejemplo:

```
function saludar(a,b)
{
	alert("Hola " + a + " y "+ b +".");
}
```

Si llamamos a esa función desde el código:

saludar("Martin","Silvia");		//Mostraría una alerta con el texto: Hola Martin y Silvia.

Los parámetros que usamos en la definición de la función a y b, no usan la palabra reservada *var* para inicializar dichas variables. Esos *parámetros a y b serán variables locales a la función*, y se inicializarán automáticamente en el momento de llamar a la función, con los valores que le pasemos en la llamada. En el siguiente apartado entraremos más en profundidad en lo que son las variables locales y globales.

Otro ejemplo de función que devuelve un valor:

```
function devolverMayor(a,b)
{
	if (a > b) then
		return a;
	else
		return b;
}
```

Ejemplo de utilización de la función anterior:

document.write ("El número mayor entre 35 y 21 es el: " + devolverMayor(35,21) + ".");

En la siguiente presentación interactiva podrás ver cómo programar una aplicación JavaScript y cómo detectar errores en el código:

CITA PARA PENSAR.

“La inteligencia es la función que adapta los medios a los fines.”

de HARTMANN, N.

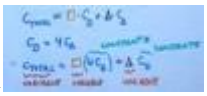
PARA SABER MÁS.

Texto enlace: Más información sobre funciones
URL: http://www.ulpgc.es/otros/tutoriales/JavaScript/cap5.htm
Título: Más información sobre funciones y ejemplos de uso.

Texto enlace: Ejemplo de funciones con parámetros
URL: http://www.desarrolloweb.com/articulos/585.php
Título: Enlace a página con ejemplos de funciones que utilizan parámetros.

3 Ámbito de las variables

Ha llegado la hora de distinguir entre las variables que se definen fuera de una función, y las que se definen dentro de las funciones.



Las variables que se definen fuera de las funciones se llaman **variables globales**.
Las variables que se definen dentro de las funciones, con la palabra reservada **var**, se llaman **variables locales**.

Una **variable global** en JavaScript tiene una connotación un poco diferente, comparando con otros lenguajes de programación. Para un script de JavaScript, el alcance de una variable global, se limita al documento actual que está cargado en la ventana del navegador o en un frame. Sin embargo cuando inicializas una variable como variable global, quiere decir que todas las instrucciones de tu script (incluidas las instrucciones que están dentro de las funciones), tendrán acceso directo al valor de esa variable. Todas las instrucciones podrán leer y modificar el valor de esa variable global.

En el momento que una página se cierra, todas las variables definidas en esa página se eliminarán de la memoria para siempre. Si necesitas que el valor de una variable persista de una página a otra, tendrás que utilizar técnicas que te permitan almacenar esa variable (como las cookies, o bien poner esa variable en el documento frameset, etc.).

Aunque el uso de la palabra reservada *var*, para inicializar variables es opcional, te recomiendo que la uses ya que así te protegerás de futuros cambios en las próximas versiones de JavaScript.

En contraste a las variables globales, una **variable local será definida dentro de una función**. Antes viste que podemos definir variables en los parámetros de una función (sin usar *var*), pero también podrás definir nuevas variables dentro del código de la función. En este caso, **si que se requiere el uso de la palabra reservada *var* cuando definimos una variable local**, ya que de otro modo, esta variable será reconocida como una variable global.

El alcance de una variable local está solamente dentro del ámbito de la función. Ninguna otra función o instrucciones fuera de la función podrán acceder al valor de esa variable.

Reutilizar el nombre de una variable global como local es uno de los bugs más sutiles y por consiguiente más difíciles de encontrar en el código de JavaScript. La variable local en momentos puntuales ocultará el valor de la variable global, sin avisarnos de ello. Como recomendación, no reutilices un nombre de variable global como local en una función, y tampoco declares una variable global dentro de una función, ya que podrás crear fallos que te resultarán difíciles de solucionar.

Ejemplo de variables locales y globales:

```
/// Uso de variables locales y globales no muy recomendable, ya que estamos empleando el mismo nombre de variable en global y en local.

var  chica  =  "Aurora";                  // variable global
var perros = "Lacky, Samba y Ronda"; // variable global

function demo()
{
        // Definimos una variable local (fíjate que es obligatorio para las variables locales usar var)
        // Esta variable local tendrá el mismo nombre que otra variable global pero con distinto contenido.
        // Si no usáramos var estaríamos modificando la variable global chica.
        var  chica  =  "Raquel";     // variable local
        document.write( "<br/>"+ perros +" no pertenecen a "+ chica +".");
}

// Llamamos a la función para que use las variables locales.
demo();

// Utilizamos las variables globales definidas al comienzo.
document.write( "<br/>"+ perros +" pertenecen a "+ chica +".");
```

Como resultado obtenemos:

```
        Lacky, Samba y Ronda no pertenecen a Raquel.
        Lacky, Samba y Ronda pertenecen a Aurora.
```

4 Funciones anidadas

Los navegadores más modernos nos proporcionan la opción de anidar unas funciones dentro de otras. Es decir podemos programar una función dentro de otra función.

Cuando no tenemos funciones anidadas, cada función que definamos será accesible por todo el código, es decir serán funciones globales. Con las funciones anidadas, podemos encapsular la accesibilidad de una función dentro de otra y hacer que esa función sea privada o local a la función principal. Tampoco te recomiendo el reutilizar nombres de funciones con esta técnica, para evitar problemas o confusiones posteriores.

La estructura de las funciones anidadas será algo así:

```
function principalA()
{
// instrucciones
        function internaA1()
        {
                // instrucciones
        }
// instrucciones
}

function principalB()
{
        // instrucciones
        function internaB1()
        {
                // instrucciones
        }

        function internaB2()
        {
                // instrucciones
        }

        // instrucciones
}
```

Una buena opción para aplicar las funciones anidadas, es cuando tenemos una secuencia de instrucciones que necesitan ser llamadas desde múltiples sitios dentro de una función, y esas instrucciones sólo tienen significado dentro del contexto de esa función principal. En otras palabras, en lugar de romper la secuencia de una función muy larga en varias funciones globales, haremos lo mismo pero utilizando funciones locales.

Ejemplo de una función anidada:

```
function hipotenusa(a, b)
{
  function cuadrado(x)
        {
                return x*x;
        }

  return Math.sqrt(cuadrado(a) + cuadrado(b));
}

document.write("<br/>La hipotenusa de 1 y 2 es: "+hipotenusa(1,2));

// Imprimirá: La hipotenusa de 1 y 2 es: 2.23606797749979
```

CITA PARA PENSAR.

“La función última de la crítica es que satisfaga la función natural de desdeñar, lo que conviene a la buena higiene del espíritu.”

de PESSOA, Fernando.

5 Funciones predefinidas del lenguaje

Has visto en unidades anteriores un montón de objetos con sus propiedades y métodos. Si te das cuenta todos los métodos en realidad son funciones (llevan siempre paréntesis con o sin parámetros). Pues bien en JavaScript, disponemos de algunos elementos que necesitan ser tratados a escala global y que no pertenecen a ningún objeto en particular (o que se pueden aplicar a cualquier objeto).

Propiedades globales en JavaScript:

Propiedad	**Descripción**
Infinity	Un valor numérico que representa el infinito positivo/negativo.
NaN	Valor que no es numérico “Not a Number”.
undefined()	Indica que a esa variable no le ha sido asignado un valor.

Te mostraremos aquí una lista de funciones predefinidas, que se pueden utilizar a nivel global en cualquier parte de tu código de JavaScript. Estas funciones no están asociadas a ningún objeto en particular. Típicamente, estas funciones te permiten convertir datos de un tipo a otro tipo.

Funciones globales o predefinidas en JavaScript:

Función	**Descripción**
decodeURI()	Decodifica los caracteres especiales de una URL excepto: , / ? : @ & = + $ #
decodeURIComponent()	Decodifica todos los caracteres especiales de una URL.
encodeURI()	Codifica los caracteres especiales de una URL excepto: , / ? : @ & = + $ #
encodeURIComponent()	Codifica todos los caracteres especiales de una URL.
escape()	Codifica caracteres especiales en una cadena, excepto: * @ - _ + . /
eval()	Evalúa una cadena y la ejecuta si contiene código u operaciones.
isFinite()	Determina si un valor es un número finito válido.
isNaN()	Determina cuando un valor no es un número.
Number()	Convierte el valor de un objeto a un número.
parseFloat()	Convierte una cadena a un número real.
parseInt()	Convierte una cadena a un entero.
unescape()	Decodifica caracteres especiales en una cadena, excepto: * @ - _ + . /

Ejemplo de la **función eval():**

```
<script type="text/javascript">
        eval("x=50;y=30;document.write(x*y)");     // Imprime 1500
        document.write("<br />" + eval("8+6"));     // Imprime 14
        document.write("<br />" + eval(x+30));      // Imprime 80
</script>
```

DEBES CONOCER.

El siguiente enlace amplía información sobre las funciones predefinidas en JavaScript.

Texto enlace: Más información y ejemplos sobre propiedades y funciones predefinidas en JavaScript.

URL: http://www.w3schools.com/jsref/jsref_obj_global.asp

Título: Enlace a la página de W3Schools, con información y ejemplos completos de cada una de las funciones predefinidas y propiedades globales en JavaScript.