UNIDAD 4 - FUNCIONES

4.1 - CONCEPTO DE FUNCIÓN

Cuando nos encontramos ante un script de una cierta dimensión o complejidad se suele dividir en partes más pequeñas, de forma que cada una permita alcanzar un determinado propósito. Para simplificar la tarea de desarrollo, los lenguajes de programación incorporan un mecanismo conocido con el nombre de funciones. Las funciones permiten agrupar bajo un identificador un conjunto de instrucciones que resuelven un problema concreto. Una función es, por tanto, un conjunto de instrucciones a las que se asigna un nombre, pueden devolver un valor y están disponibles para su ejecución tantas veces como se desee con solo invocarlas a través del identificador que las representa. Se pueden definir tantas funciones como sea preciso. A la hora de trabajar con funciones debemos distinguir dos aspectos: la creación de la función (o definición) y la utilización de la función (o llamada). En el código que aparece a continuación se puede observar la repetición de un conjunto de operaciones para diferentes valores. Esta situación es un claro ejemplo que puede resolverse eficazmente con funciones.

```
var resultado;
var numero1 = 4;
var numero2 = 2;
// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 7;
numero2 = 1;
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 8;
numero2 = 2;
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
```

4.1.1 -CREACIÓN DE UNA FUNCIÓN.

Para crear una función debemos determinar los siguientes aspectos:

- El nombre de la función.
- Los parámetros o valores sobre los que actuará, es decir, los argumentos.
- Las acciones que deberá realizar.
- El valor que devolverá. Aunque puede no devolver ningún valor.

Y aplicar el siguiente formato:

```
function nombredefuncion (listadeargumentos)
{
        instrucciones...
        ...
        [return valorderetorno;]
}
```

Donde:

- nombredefuncion es un identificador válido que servirá para realizar llamadas a la función.
- listadeargumentos es una lista de variables (separadas por comas) que recogerán los valores que se pasen en la llamada a la función.
- valorderetorno es una expresión cuyo valor devolverá la función.

El siguiente ejemplo define una función llamada suma que recibe dos valores y devuelve su suma:

```
function suma(A,B)
{
        var C;
        C = A + B;
        return C;
}
```

Cuando el navegador encuentra la definición de una función, carga dicha función en la memoria pero no la ejecutará hasta que se produzca una llamada a la función.

4.1.2 -UBICACIÓN DE LA DEFINICIÓN DE UNA FUNCIÓN.

Las funciones pueden declararse en cualquier parte de una página HTML entre las etiquetas **<SCRIPT>** y **</SCRIPT>** pero teniendo en cuenta que no pueden definirse dentro de otra función ni dentro de una estructura de control. Normalmente se definen dentro de la cabecera de una página HTML, así cuando se carga la página en el navegador del cliente estarán disponibles para ser utilizadas. En el ejemplo siguiente se declara una función pero la misma no se utiliza y por lo tanto el programa aparentemente no hace nada.

```
<html>
<head>
<title>Funciones</title>
<script language="JavaScript">
function Saludo()
{
        document.write("hola mundo");
}
</script>
</head>
<body>
</body>
</html>
```

Ejemplo 20 – Declaración de una función

4.1.3 -LLAMADA A UNA FUNCIÓN.

Una vez definida la función podremos hacer llamadas que producirán la ejecución de la misma como si se tratase de cualquier instrucción. Estas llamadas deben hacerse siguiendo el formato:

```
nombredefuncion(listadeparámetros)
```

Debemos tener en cuenta que la llamada puede devolver un valor y que debemos hacer algo con él como escribirlo, asignarlo a una variable, etcétera. Por ejemplo, para hacer una llamada a la función suma y escribir el valor devuelto en el documento actual escribiremos:

```
document.write(suma(2,3));
```

```
<html>
<head>
<title>Llamada a una función</title>
<script language="JavaScript">
function Hora()
{
        var hoy =new Date()
        document.write(hoy.getHours(),":",hoy.getMinutes())
}
</script>
</head>
<body >
<script language="JavaScript">
        document.write("Hora actual: ");
        Hora();
</script>
</body>
</html>
```

Ejemplo 21 – Llamada a una función

4.2 - FUNCIONES CON ARGUMENTOS

Los argumentos son variables locales a la función que toman un valor determinado cuando se les invoca. Las funciones con argumentos generan resultados distintos dependiendo de los valores que tomen los argumentos cuando se llama a la función. La función suma expuesta a continuación recibe dos valores y devuelve su suma:

```
<html>
<head>
<title>Funciones con argumentos</title>
<script language="JavaScript">
function Suma(a,b)
{
        var resultado;
        resultado=a+b;
        return (resultado);
}
</script>
</head>
<body >
<script language="JavaScript">
var S, numero1,numero2;
numero1=parseInt(prompt("Introduzca un numero",0)); // parseInt se explica más adelante
numero2=parseInt(prompt("Introduzca otro numero",0));
S=Suma(numero1,numero2);
alert ("La suma es: "+S);
</script>
</body>
</html>
```

Ejemplo 22 – Funciones con argumentos (versión 1)

En el ejemplo siguiente se puede ver otro enfoque diferente de la función Suma, ya que en este caso no sólo se calcula la suma sino que se muestra en resultado por pantalla desde la misma función. Se puede comprobar que una misma función se puede llamar varias veces desde el mismo programa.

```
<HTML>
<HEAD>
<SCRIPT LANGUAJE="JavaScript">
function suma(A,B)
{
        var resultado;
        resultado = A + B;
        alert("El resultado es " + resultado);
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
suma (4 , 2);
suma (7 , 1);
suma (8 , 2);
</SCRIPT>
</BODY>
</HTML>
```

Ejemplo 23 – Funciones con argumentos (versión 2)

4.3 - CUESTIONES IMPORTANTES EN EL USO DE FUNCIONES.

- Tanto las funciones como las llamadas deben ir entre etiquetas **<SCRIPT> </SCRIPT>**
- La cláusula ***return*** produce la salida de la función devolviendo (opcionalmente) un valor; como ya hemos visto en el ejemplo anterior.
- JavaScript permite escribir varias cláusulas return en una misma función.
- Se pueden escribir funciones que no devuelven ningún valor sino que solamente realizan una o varias acciones. También se pueden escribir funciones que no tengan parámetros.
- Si al llamar a una función se le pasa un número menor de parámetros, los parámetros que no han obtenido valor como resultado de la llamada quedarán con el valor null. Por ejemplo, si la llamada a la función suma se realiza con un sólo número el resultado será: NaN
- Para evitar este tipo de situaciones podemos comprobar el número de parámetros que se ha pasado a una función mediante arguments.length tal como podemos apreciar en el siguiente ejemplo.

```
function suma(A,B)
{
        var C;
        if (arguments.length < 2)
        {
                B = 0;
        }
        if (arguments.length < 1)
        {
                A = 0;
        }
        C = A + B;
        return C;
}
```

4.4 - VARIABLES GLOBALES Y LOCALES

Son variables locales aquellas que se declaran en la definición de la función y sólo son accesibles en el ámbito de la función donde se declaran. Las variables globales se declaran fuera de la declaración de cualquier función y se puede tener acceso a ellas desde cualquier parte del documento.

```
<HTML>
<HEAD>
<SCRIPT LANGUAJE="JavaScript">
var C; //variable global, la puede usar cualquier función
C=0;
function suma(A,B)
{
        C = A + B;
        return C;
}
function resta(A,B)
{
        return C+A-B;
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
document.write("Sumamos dos valores: ");
document.write(suma(2,3));
document.write('<BR>');
document.write("Restamos dos valores: ");
document.write(resta(10,20));
</SCRIPT>
</BODY>
</HTML>
```

Ejemplo 24 – Variables locales y globales

4.5 - FUNCIONES PREDEFINIDAS

Existen algunas funciones propias del lenguaje que permitir realizar operaciones comunes.

eval (cadena). Esta función permite que una cadena cuyo contenido debe ser código Javascript pueda ser evaluada por el intérprete de Javascript. No es una función destinada a los programadores menos experimentados ya que la potencia que encierra implica tener ciertos conocimientos de programación.

isNaN(argumento). Es una función boolena (su resultado es true o false). La función devuelve true si el argumento NO es un número y false en caso contrario.

parseInt (cadena[,base]). La función convierte una cadena de caracteres a su valor numérico. Opcionalmente se puede establecer la base para indicar que la cadena de caracteres representa un número en la base señalada. Si no se indica la base, el sistema supone que se trata de la base 10. La función finalizará la conversión cuando encuentre

un carácter que no sea un dígito del sistema cuya base haya sido especificada. Si la base es 10 serán los dígitos (0..9), en base octal (0..7), en base hexadecimal (0..9,A..F).

parseFloat (cadena). La función convierte una cadena de caracteres a su valor en coma flotante, es decir a un número real expresado en notación exponencial. La función finalizará la conversión cuando encuentre un carácter que no sea un dígito (0..9), un punto decimal (.) o una "e".

escape (cadena). Codifica la cadena sustituyendo todos los caracteres que no forman parte de la codificación ASCII1 de 7 bits a sus secuencia de códigos de escape asociada. Estos códigos serán devueltos con el formato %NN, donde NN es el código de escape en hexadecimal del carácter.

unescape (cadena). Realiza el proceso inverso al anterior, es decir devuelve la cadena formada por los caracteres cuyo código le especificaremos. Los códigos deben ser especificados con el mismo formato con el que escape los proporciona: %NN, donde NN es el código en hexadecimal.

toString (argumento). Convierte a cadena el dato especificado en argumento. Normalmente se utiliza para convertir números a cadena pero también funciona conotros tipos como Booleano. Su utilización es restringida ya que JavaScript aplica una conversión automática de tipos a cadena cuando encuentra expresiones de concatenación en las que hay diversos tipos de datos.

typeof(argumento). Devuelve una cadena que indica el tipo del argumento (number, string, boolean, undefined).

Ejemplo	Resultado
eval('alert ("Hola");');	Muestra una ventana con el texto Hola
isNaN(12);	False
isNaN("Hola");	True
parseInt("27.11.2000");	27

[1] http://es.wikipedia.org/wiki/ASCII

parseInt("20H30M07S");	20
parseInt("345.6e5");	345
parseInt("VUELO506");	NaN (Not a Number)
parseInt("27", 8);	23
parseInt("11111111", 2);	255
parseInt("A2", 16);	162
parseFloat("27.11.2000");	27.11
parseFloat("20H30M07S");	20
parseFloat("345.6e5");	34560000
parseFloat("VUELO506");	NaN (Not a Number)
escape ("Buenos días");	Buenos d%EDas
unescape ("Buenos d%EDas");	Buenos días
Typeof ("Hola");	string