Indice:

Funciones - Continuación

Las ideas que hemos explicado anteriormente sobre funciones no son las únicas que debemos aprender para manejarlas en toda su potencia. Las funciones también tienen una entrada y una salida de datos. En este artículo veremos cómo podemos enviar datos a las funciones Javascript.

Parámetros

Los parámetros se usan para mandar valores a las funciones. Una función trabajará con los parámetros para realizar las acciones. Por decirlo de otra manera, los parámetros son los valores de entrada que recibe una función. Por poner un ejemplo sencillo de entender, una función que realizase una suma de dos números tendría como parámetros a esos dos números. Los dos números son la entrada, así como la salida sería el resultado de la suma, pero eso lo veremos más tarde.

Veamos un ejemplo anterior en el que creábamos una función para mostrar un mensaje de bienvenida en la página web, pero al que ahora le vamos a pasar un parámetro que contendrá el nombre de la persona a la que hay que saludar.

```
function escribirBienvenida(nombre){
      document.write("<H1>Hola " + nombre + "</H1>")
}
```

Como podemos ver en el ejemplo, para definir en la función un parámetro tenemos que poner el nombre de la variable que va a almacenar el dato que le pasemos. Esa variable, que en este caso se llama nombre, tendrá como valor el dato que le pasemos a la función cuando la llamemos. Además, la variable donde recibimos el parámetro tendrá vida durante la ejecución de la función y dejará de existir cuando la función termine su ejecución.

Para llamar a una función que tiene parámetros se coloca entre paréntesis el valor del parámetro. Para llamar a la función del ejemplo habría que escribir:

```
escribirBienvenida("Alberto García")
```

Al llamar a la función así, el parámetro nombre toma como valor "Alberto García" y al escribir el saludo por pantalla escribirá "Hola Alberto García" entre etiquetas <H1>.

Los parámetros pueden recibir cualquier tipo de datos, numérico, textual, boleano o un objeto. Realmente no especificamos el tipo del parámetro, por eso debemos tener un cuidado especial al definir las acciones que realizamos dentro de la función y al pasarle valores, para asegurarnos que todo es consecuente con los tipos de datos que esperamos tengan nuestras variables o parámetros.

Múltiples parámetros

Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los nombres de los parámetros separados por comas, dentro de los paréntesis. Veamos rápidamente la sintaxis para que la función de antes, pero hecha para que reciba dos parámetros, el primero el nombre al que saludar y el segundo el color del texto.

```
function escribirBienvenida(nombre,colorTexto){
	document.write("<FONT color='" + colorTexto + "'>")
	document.write("<H1>Hola " + nombre + "</H1>")
	document.write("</FONT>")
}
```

Llamaríamos a la función con esta sintaxis. Entre los paréntesis colocaremos los valores de los parámetros.

```
var miNombre = "Pepe"
var miColor = "red"
escribirBienvenida(miNombre,miColor)
```

He colocado entre los paréntesis dos variables en lugar de dos textos entrecomillados. Cuando colocamos variables entre los parámetros en realidad lo que estamos pasando a la función son los valores que contienen las variables y no las mismas variables.

Los parámetros se pasan por valor

Al hilo del uso de parámetros en nuestros programas Javascript, tenemos que saber que los parámetros de las funciones se pasan por valor. Esto quiere decir que estamos pasando valores y no variables. En la práctica, aunque modifiquemos un parámetro en una función, la variable original que habíamos pasado no cambiará su valor. Se puede ver fácilmente con un

ejemplo.

```
function pasoPorValor(miParametro){
      miParametro = 32
      document.write("he cambiado el valor a 32")
}
var miVariable = 5
pasoPorValor(miVariable)
document.write ("el valor de la variable es: " + miVariable)
```

En el ejemplo tenemos una función que recibe un parámetro y que modifica el valor del parámetro asignándole el valor 32. También tenemos una variable, que inicializamos a 5 y posteriormente llamamos a la función pasándole esta variable como parámetro. Como dentro de la función modificamos el valor del parámetro podría pasar que la variable original cambiase de valor, pero como los parámetros no modifican el valor original de las variables, ésta no cambia de valor.
De este modo, una vez ejecutada la función, al imprimir en pantalla el valor de miVariable se imprimirá el número 5, que es el valor original de la variable, en lugar de 32 que era el valor con el que habíamos actualizado el parámetro.

En Javascript sólo se pueden pasar las variables por valor.

Devolución de datos

Una función puede devolver datos hacia afuera por medio de la expresión **return**.
Naturalmente, podemos devolver cualquier tipo de datos. Sin embargo hay que tener en cuenta una serie de cuestiones:
Siempre se devuelven objetos, como ya hemos visto, y por lo tanto podemos devolver un objeto creado en la misma función. Normalmente, cuando creamos una variable dentro de una función, esta variable existe sólo para esa función, y desaparece en el momento en que la función termina (la variable se encuentra en la pila de memoria, y cuando la función desaparece, también lo hace la pila); pero en el caso de que devolvamos el objeto, no se devuelve exactamente la misma variable, si no que se devuelve su contenido. Cuando devolvemos true ó un valor distinto que cero, para JavaScript es lo mismo, y si devolvemos false o 0, también viene a ser lo mismo. Esta es una regla estándar para muchos lenguajes como JavaScript, Java, PHP, Perl, etc… No es preciso que una función devuelva nada. No es necesario usar return. Además, también es posible que en vez de devolver resultados, se modifiquen variables globales, es decir, variables creadas fuera de la función y que se usan dentro. Si queremos salir de una función antes de tiempo, porque algo ha fallado o no hay nada que hacer en un caso específico, podemos simplemente escribir “return;”, lo que nos permitirá salir sin más y no devolver ningún valor.
Estas consideraciones son importantes a nivel general y es importante tenerlas en cuenta. Vamos a ver como funcionan con algunos ejemplos:

```
function dev_variable()
 {

      variable = true;
      return variable;

 }

 var var1 = dev_variable();
```

Como vemos, hemos declarado una variable local a la función y la hemos devuelto, pero solo se devuelve realmente el valor. Esto pasa en todos los casos (Nota técnica: cuando se devuelve un objeto, se devuelven sus datos en forma de objeto de esa clase; esto lo entenderemos mejor en el capítulo siguiente). Veamos este otro ejemplo:

```
function dev_true()
       {
            return true;
       }

 if (dev_true())
       {
            alert("es true");
       }

 if (true)
       {
           alert("también es true");
       }

 if (0)
       {
            alert("este también es true");
       }
```

Como hemos enunciado antes, un valor true y una valor distinto de cero son siempre verdad. En el ultimo caso, no se ejecutará porque es cero, y por lo tanto, falso.
Por último, veamos como salir de una función sin necesidad de devolver nada en cualquier momento:

```
function salir()
{
       document.write("hola");
       document.write("que pasa");
       return;
       alert("adiós");
 }

 salir();
```

En este ejemplo, la última linea no se ejecutará nunca porque hemos salido sin más en la linea anterior.

Strings - Métodos y propiedades

En javascript las variables de tipo texto son objetos de la clase String. Esto quiere decir que cada una de las variables que creamos de tipo texto tienen una serie de propiedades y métodos. Recordamos que las propiedades son las características, como por ejemplo longitud en caracteres del string y los métodos son funcionalidades, como pueden ser extraer un substring o poner el texto en mayúsculas.
Para crear un objeto de la clase String lo único que hay que hacer es asignar un texto a una variable. El texto va entre comillas, como ya hemos visto en los capítulos de sintaxis. También se puede crear un objeto string con el operador new, que veremos más adelante. La única diferencia es que en versiones de Javascript 1.0 no funcionará new para crear los Strings.

Propiedades de String

Length
La clase String sólo tiene una propiedad: length, que guarda el número de caracteres del String.

Métodos de String

Los objetos de la clase String tienen una buena cantidad de métodos para realizar muchas cosas interesantes. Primero vamos a ver una lista de los métodos más interesantes y luego vamos a ver otra lista de métodos menos útiles.
charAt(indice)

Devuelve el carácter que hay en la posición indicada como índice. Las posiciones de un string empiezan en 0.

indexOf(carácter,desde)

Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un string. Si no encuentra el carácter en el string devuelve -1. El segundo parámetro es opcional y sirve para indicar a partir de que posición se desea que empiece la búsqueda.

lastIndexOf(carácter,desde)

Busca la posición de un carácter exáctamente igual a como lo hace la función indexOf pero desde el final en lugar del principio. El segundo parámetro indica el número de caracteres desde donde se busca, igual que en indexOf.

replace(substring_a_buscar,nuevoStr)

Implementado en Javascript 1.2, sirve para reemplazar porciones del texto de un string por otro texto, por ejemplo, podríamos uilizarlo para reemplazar todas las apariciones del substring "xxx" por "yyy". El método no reemplaza en el string, sino que devuelve un resultante de hacer ese reemplazo. Acepta expresiones regulares como substring a buscar.

split(separador)

Este método sólo es compatible con javascript 1.1 en adelante. Sirve para crear un vector a partir de un String en el que cada elemento es la parte del String que está separada por el separador indicado por parámetro.

substring(inicio,fin)

Devuelve el substring que empieza en el carácter de inicio y termina en el carácter de fin. Si intercambiamos los parámetros de inicio y fin también funciona. Simplemente nos da el substring que hay entre el carácter menor y el mayor.

toLowerCase()

Pone todas los caracteres de un string en minúsculas.

toUpperCase()

Pone todas los caracteres de un string en mayúsculas.

toString()

Este método lo tienen todos los objetos y se usa para convertirlos en cadenas.

Hasta aquí hemos visto los métodos que nos ayudarán a tratar cadenas. Ahora vamos a ver otros métodos que son menos útiles, pero hay que indicarlos para que quede constancia de ellos. Todos sirven para aplicar estilos a un texto y es como si utilizásemos etiquetas HTML. Veamos cómo.

Estructuras de Control : switch - case

Las estructuras de control son la manera con la que se puede dominar el flujo de los programas, para hacer cosas distintas en función de los estados de las variables. En el Manual de Javascript ya empezamos a ver las estructuras de control y ahora le ha tocado el turno a SWITCH, una estructura un poco más compleja que permite hacer múltiples operaciones dependiendo del estado de una variable.

En este artículo veremos que switch nos sirve para tomar decisiones en función de distintos estados de las variables. Esta expresión se utiliza cuando tenemos múltiples posibilidades

como resultado de la evaluación de una sentencia.

La estructura SWITCH se incorporó a partir de la versión 1.2 de Javascript (Netscape 4 e Internet Explorer 4). Su sintaxis es la siguiente.

```
switch (expresión) {
        case valor1:
                Sentencias a ejecutar si la expresión tiene como valor a valor1
                break
        case valor2:
                Sentencias a ejecutar si la expresión tiene como valor a valor2
                break
        case valor3:
                Sentencias a ejecutar si la expresión tiene como valor a valor3
                break
        default:
                Sentencias a ejecutar si el valor no es ninguno de los anteriores
}
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como deseemos. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra break es opcional, pero si no la ponemos a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en nuestro esquema anterior no hubiese ningún break y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y default.

También es opcional la opción default u opción por defecto.

Veamos un ejemplo de uso de esta estructura. Supongamos que queremos indicar que día de la semana es. Si el día es 1 (lunes) sacar un mensaje indicándolo, si el día es 2 (martes) debemos sacar un mensaje distinto y así sucesivamente para cada día de la semana, menos en el 6 (sábado) y 7 (domingo) que queremos mostrar el mensaje "es fin de semana". Para días mayores que 7 indicaremos que ese día no existe.

```
switch (dia_de_la_semana) {
      case 1:
                  document.write("Es Lunes")
                  break
```

```
      case 2:
                document.write("Es Martes")
                break
      case 3:
                document.write("Es Miércoles")
                break
      case 4:
                document.write("Es Jueves")
                break
      case 5:
                document.write("Es viernes")
                break
      case 6:
      case 7:
                document.write("Es fin de semana")
                break
      default:
                document.write("Ese día no existe")
}
```

El ejemplo es relativamente sencillo, solamente puede tener una pequeña dificultad, consistente en interpretar lo que pasa en el caso 6 y 7, que habíamos dicho que teníamos que mostrar el mismo mensaje. En el caso 6 en realidad no indicamos ninguna instrucción, pero como tampoco colocamos un break se ejecutará la sentencia o sentencias del caso siguiente, que corresponden con la sentencia indicada en el caso 7 que es el mensaje que informa que es fin de semana. Si el caso es 7 simplemente se indica que es fin de semana, tal como se pretendía.

DOM - Create Element - Append Child - setAttribute - removeChild

Uso: **var elemento = document.createElement(tag);**
Con éste método del objeto document crearíamos un nuevo elemento con un tag determinado.
Ejemplo: var div = document.createElement('DIV');

Uso: **elemento1.appendChild(elemento2);**
Con este método añadiremos el elemento “elemento2” a “elemento1”. En el siguiente ejemplo, añadiremos elementos a una lista de forma dinámica:

```
<input type=”text” id=”texto” />
<input type=”button” value=”Crear” onclick=”crear()” />
<ul id=”lista”></ul>
<script type=”text/javascript”>
function crear() {
```

```
// Obtenemos el valor entrado en la caja de texto
var valor = document.getElementById("texto").value;
// Creamos un nuevo elemento LI
var li = document.createElement("LI");
// Añadimos el valor introducido al nuevo elemento
li.innerHTML = valor;
// Añadimos el elemento LI a la lista UL
var ul = document.getElementById("UL");
ul.appendChild(li);
// Vaciamos la caja de texto
document.getElementById("texto").value = "";
}
</script>
```

removeChild()
Uso: elemento.removeChild(hijo);
Este método es el usado para eliminar elementos. Se elimina el elemento hijo del objeto. Si queremos eliminar un objeto concreto, tendremos que hacerlo de la siguiente manera:
// Obtenemos el elemento
var el = document.getElementById("elemento-a-eliminar");
// Obtenemos el padre de dicho elemento
// con la propiedad "parentNode"
var padre = el.parentNode;
// Eliminamos el hijo (el) del elemento padre
padre.removeChild(el);

SetAttribute
Con javascript podemos agregar o quitar un atributo a un campo HTML. Por ejemplo al cliquear sobre un checkbox podemos hacer que otro campo se habilite o deshabilite, esto es a veces util en los formularios. Definimos una funcion js:

```
function changeState(check, element){
 if(!check.checked){
       element.setAttribute('disabled','disabled'); //atributo y valor
    }else{
      element.removeAttribute('disabled'); //el atributo
    }
}
```

Y luego en nuestro ducumento HTML definimos el evento onclick (por ejemplo) asi::

```
<input onclick="changeState(this,document.getElementById('nombre'))"
```

```
type="checkbox" />
<input id="nombre" disabled="disabled" type="text" /
```

Lo que produce es que al dar clic sobre el checkbox el campo input se habilite o deshabilite.