

JavaScript

CONTENIDO:

Árbol de nodos	2
Seleccionando nodos	4
Creación y eliminación de nodos	6
Acceso directo a los atributos	7
Artículo interior	4
Artículo interior	5
Artículo interior	6



Unidad 6 - Funciones

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de comercio electrónico por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.

Cuando una serie de instrucciones se repiten una y otra vez, se complica demasiado el código fuente de la aplicación, ya que:

El código de la aplicación es mucho más largo porque muchas instrucciones están repetidas.

Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que se convierte en un trabajo muy pesado y muy propenso a cometer errores.

Las funciones son la solución a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación. Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

1.— Concepto de función

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

En el siguiente ejemplo, las instrucciones que suman los dos números y muestran un mensaje con el resultado se repiten una y otra vez:

```
var resultado;
var numero1 = 3;
var numero2 = 5;
// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
numero1 = 10;
numero2 = 7;
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
.....
```

Aunque es un ejemplo muy sencillo, parece evidente que repetir las mismas instrucciones a lo largo de todo el código no es algo recomendable. La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo "en este punto, se ejecutan las instrucciones que se han extraído":

```
var resultado;
var numero1 = 3;
var numero2 = 5;
/* En este punto, se llama a la función que suma 2 números y muestra el resultado */
numero1 = 10;
numero2 = 7;
/* En este punto, se llama a la función que suma 2 números y muestra el resultado */
numero1 = 5;
numero2 = 8;
/* En este punto, se llama a la función que suma
2 números y muestra el resultado */
```

El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM

Para que la solución del ejemplo anterior sea válida, las instrucciones comunes se tienen que agrupar en una función a la que se le puedan indicar los números que debe sumar antes de mostrar el mensaje.

Por lo tanto, en primer lugar se debe crear la función básica con las instrucciones comunes. Las funciones en JavaScript se definen mediante la palabra reservada `function`, seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion() {
  ...
}
```

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código. Después del nombre de la función, se incluyen dos paréntesis cuyo significado se detalla más adelante. Por último, los símbolos `{ }` se utilizan para encerrar todas las instrucciones que pertenecen a la función (de forma similar a como se encierran las instrucciones en las estructuras `if` o `for`).

Volviendo al ejemplo anterior, se crea una función llamada `suma_y_muestra` de la siguiente forma:

```
function suma_y_muestra() {
  resultado = numero1 + numero2;
  alert("El resultado es " + resultado);
}
```

Aunque la función anterior está correctamente creada, no funciona como debería ya que le faltan los "argumentos", que se explican en la siguiente sección. Una vez creada la función, desde cualquier punto del código se puede llamar a la función para que se ejecuten sus instrucciones (además de "llamar a la función", también se suele utilizar la expresión "invocar a la función").

La llamada a la función se realiza simplemente indicando su nombre, incluyendo los paréntesis del final y el carácter `;` para terminar la instrucción:

```
function suma_y_muestra() {
  resultado = numero1 + numero2;
```

```
  alert("El resultado es " + resultado);
}
var resultado;
var numero1 = 3;
var numero2 = 5;
suma_y_muestra();
numero1 = 10;
numero2 = 7;
suma_y_muestra();
numero1 = 5;
numero2 = 8;
suma_y_muestra();
...
```

El código del ejemplo anterior es mucho más eficiente que el primer código que se mostró, ya que no existen instrucciones repetidas. Las instrucciones que suman y muestran mensajes se han agrupado bajo una función, lo que permite ejecutarlas en cualquier punto del programa simplemente indicando el nombre de la función.

Lo único que le falta al ejemplo anterior para funcionar correctamente es poder indicar a la función los números que debe sumar. Cuando se necesitan pasar datos a una función, se utilizan los "argumentos", como se explica en la siguiente sección.

2.— Argumentos y valores de retorno

Las funciones más sencillas no necesitan ninguna información para producir sus resultados. Sin embargo, la mayoría de funciones de las aplicaciones reales deben acceder al valor de algunas variables para producir sus resultados.

Las variables que necesitan las funciones se llaman argumentos. Antes de que pueda utilizarlos, la función debe indicar cuántos argumentos necesita y cuál es el nombre de cada argumento. Además, al invocar la función, se deben incluir los valores que se le van a pasar a la función. Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (,).

Siguiendo el ejemplo anterior, la función debe indicar que necesita dos argumentos, correspondientes a los dos números que tiene que sumar:

```
function suma_y_muestra(primerNumero, segundoNumero)
{ ... }
```

A continuación, para utilizar el valor de los argumentos dentro de la función, se debe emplear el mismo nombre con el que se definieron los argumentos:

```
function suma_y_muestra(primerNumero, segundoNumero)
{ ... }

    var resultado = primerNumero + segundoNumero;

    alert("El resultado es " + resultado);
}
```

Dentro de la función, el valor de la variable primerNumero será igual al primer valor que se le pase a la función y el valor de la variable segundoNumero será igual al segundo valor que se le pasa. Para pasar valores a la función, se incluyen dentro de los paréntesis utilizados al llamar a la función:

```
// Definición de la función

function suma_y_muestra(primerNumero, segundoNumero)
{ ... }

    var resultado = primerNumero + segundoNumero;

    alert("El resultado es " + resultado);
```

```
}

// Declaración de las variables

var numero1 = 3;

var numero2 = 5;

// Llamada a la función

suma_y_muestra(numero1, numero2);
```

En el código anterior, se debe tener en cuenta que:

- Aunque casi siempre se utilizan variables para pasar los datos a la función, se podría haber utilizado directamente el valor de esas variables: `suma_y_muestra(3, 5);`
- El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios.
- El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función; el segundo valor indicado en la llamada, es el segundo valor que espera la función y así sucesivamente.
- Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.
- No es obligatorio que coincida el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan. En el ejemplo anterior, los argumentos que se pasan son `numero1` y `numero2` y los argumentos que utiliza la función son `primerNumero` y `segundoNumero`.

A continuación se muestra otro ejemplo de una función que calcula el precio total de un producto a partir de su precio básico:

```
// Definición de la función
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
}
```

// Llamada a la función

```
calculaPrecioTotal(23.34);
```

La función anterior toma como argumento una variable llamada precio y le suma los impuestos y los gastos de envío para obtener el precio total. Al llamar a la función, se pasa directamente el valor del precio básico mediante el número 23.34.

No obstante, el código anterior no es demasiado útil, ya que lo ideal sería que la función pudiera devolver el resultado obtenido para guardarlo en otra variable y poder seguir trabajando con este precio total:

```
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
}
```

// El valor devuelto por la función, se guarda en una variable

```
var precioTotal = calculaPrecioTotal(23.34);
```

// Seguir trabajando con la variable "precioTotal"

Afortunadamente, las funciones no solamente puede recibir variables y datos, sino que también pueden devolver los valores que han calculado. Para devolver valores dentro de una función, se utiliza la palabra reservada return. Aunque las funciones pueden devolver valores de cualquier tipo, solamente pueden devolver un valor cada vez que se ejecutan.

```
function calculaPrecioTotal(precio) {
```

```
  var impuestos = 1.16;
```

```
  var gastosEnvio = 10;
```

```
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
```

```
  return precioTotal;
```

```
}
```

```
var precioTotal = calculaPrecioTotal(23.34);
```

// Seguir trabajando con la variable "precioTotal"

Para que la función devuelva un valor, solamente es necesario escribir la palabra reservada return junto con el nombre de la variable que se quiere devolver. En el ejemplo anterior, la ejecución de la función llega a la instrucción return precioTotal; y en ese momento, devuelve el valor que contenga la variable precioTotal.

Como la función devuelve un valor, en el punto en el que se realiza la llamada, debe indicarse el nombre de una variable en el que se guarda el valor devuelto:

```
var precioTotal = calculaPrecioTotal(23.34);
```

Si no se indica el nombre de ninguna variable, JavaScript no muestra ningún error y el valor devuelto por la función simplemente se pierde y por tanto, no se utilizará en el resto del programa. En este caso, tampoco es obligatorio que el nombre de la variable devuelta por la función coincida con el nombre de la variable en la que se va a almacenar ese valor.

Si la función llega a una instrucción de tipo return, se devuelve el valor indicado y finaliza la ejecución de la función. Por tanto, todas las instrucciones que se incluyen después de un return se ignoran y por ese motivo la instrucción return suele ser la última de la mayoría de funciones.

Para que el ejemplo anterior sea más completo, se puede añadir otro argumento a la función que indique el porcentaje de impuestos que se debe añadir al precio del producto. Evidentemente, el nuevo argumento se debe añadir tanto a la definición de la función como a su llamada:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
```

```
  var gastosEnvio = 10;
```

```

var precioConImpuestos = (1 + porcentajImpues-
tos/100) * precio;

var precioTotal = precioConImpuestos + gastosEnvio;

return precioTotal;
}

var precioTotal = calculaPrecioTotal(23.34, 16);

var otroPrecioTotal = calculaPrecioTotal(15.20, 4);

```

Para terminar de completar el ejercicio anterior, se puede redondear a dos decimales el precio total devuelto por la función:

```

function calculaPrecioTotal(precio, porcentajImpuestos) {
    var gastosEnvio = 10;

    var precioConImpuestos = (1 + porcentajImpuestos/100) *
precio;

    var precioTotal = precioConImpuestos + gastosEnvio;

    return precioTotal.toFixed(2);
}

var precioTotal = calculaPrecioTotal(23.34, 16);

```

3.— Funciones predefinidas

JavaScript incorpora una serie de funciones predefinidas para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las funciones que ofrece JavaScript.

Funciones útiles para cadenas de texto

A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas de texto:

length, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

```

var mensaje = "Hola Mundo";

var numeroLetras = mensaje.length; // numeroLetras =
10

```

El operador + se emplea para concatenar varias cadenas de texto

```

var mensaje1 = "Hola";

var mensaje2 = " Mundo";

var mensaje = mensaje1 + mensaje2; // mensaje = "Hola
Mundo"

```

Además del operador +, también se puede utilizar la función concat()

```

var mensaje1 = "Hola";

var mensaje2 = mensaje1.concat(" Mundo"); // mensaje2 =
"Hola Mundo"

```

Las cadenas de texto también se pueden unir con variables numéricas:

```

var variable1 = "Hola ";

var variable2 = 3;

var mensaje = variable1 + variable2; // mensaje = "Hola 3"

```

Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras:

```

var mensaje1 = "Hola";

var mensaje2 = "Mundo";

var mensaje = mensaje1 + mensaje2; // mensaje =
"HolaMundo"

```

Los espacios en blanco se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

```

var mensaje1 = "Hola";

var mensaje2 = "Mundo";

var mensaje = mensaje1 + " " + mensaje2; // mensaje = "Hola
Mundo"

```

toUpperCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toUpperCase(); // mensaje2 = "HOLA"
```

toLowerCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toLowerCase(); // mensaje2 = "hola"
```

charAt(posicion), obtiene el carácter que se encuentra en la posición indicada:

```
var mensaje = "Hola";
var letra = mensaje.charAt(0); // letra = H
letra = mensaje.charAt(2); // letra = l
```

indexOf(character), calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
var mensaje = "Hola";
var posicion = mensaje.indexOf('a'); // posicion = 3
posicion = mensaje.indexOf('b'); // posicion = -1
```

Su función análoga es **lastIndexOf()**:

lastIndexOf(character), calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
var mensaje = "Hola";
var posicion = mensaje.lastIndexOf('a'); // posicion = 3
posicion = mensaje.lastIndexOf('b'); // posicion = -1
```

La función lastIndexOf() comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

substring(inicio, final), extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5); // porcion = "Mundo"
porcion = mensaje.substring(7); // porcion = "ndo"
```

Si se indica un inicio negativo, se devuelve la misma cadena original:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(-2); // porcion = "Hola Mundo"
```

Cuando se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final (es decir, la posición inicio está incluida y la posición final no):

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(1, 8); // porcion = "ola Mun"
porcion = mensaje.substring(3, 4); // porcion = "a"
```

Si se indica un final más pequeño que el inicio, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al inicio y el más grande al final:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(5, 0); // porcion = "Hola "
porcion = mensaje.substring(0, 5); // porcion = "Hola "
```

split(separador), convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado:

```
var mensaje = "Hola Mundo, soy una cadena de texto!";
var palabras = mensaje.split(" ");
```

```
// palabras = ["Hola", "Mundo,", "soy", "una", "cadena",
"de", "texto!"];
```

Con esta función se pueden extraer fácilmente las letras que forman una palabra:

```
var palabra = "Hola";
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

4.— Funciones para Arrays

A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:

length, calcula el número de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];
var numeroVocales = vocales.length; // numeroVocales = 5
```

concat(), se emplea para concatenar los elementos de varios arrays

```
var array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6); // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5, 6]
```

join(separador), es la función contraria a split(). Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado

```
var array = ["hola", "mundo"];
var mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" "); // mensaje = "hola mundo"
```

pop(), elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
var array = [1, 2, 3];
var ultimo = array.pop();
// ahora array = [1, 2], ultimo = 3
```

push(), añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento.

(También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.push(4);
// ahora array = [1, 2, 3, 4]
```

shift(), elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.

```
var array = [1, 2, 3];
var primero = array.shift();
// ahora array = [2, 3], primero = 1
```

unshift(), añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.unshift(0);
// ahora array = [0, 1, 2, 3]
```

reverse(), modifica un array colocando sus elementos en el orden inverso a su posición original:

```
var array = [1, 2, 3];
array.reverse();
// ahora array = [3, 2, 1]
```


4.— Funciones para Arrays

A continuación se muestran algunas de las funciones y propiedades más útiles para el manejo de números.

NaN, (del inglés, "Not a Number") JavaScript emplea el valor NaN para indicar un valor numérico no definido (por ejemplo, la división 0/0).

```
var numero1 = 0;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor NaN
```

isNaN(), permite proteger a la aplicación de posibles valores numéricos no definidos

```
var numero1 = 0;
var numero2 = 0;
if(isNaN(numero1/numero2)) {
    alert("La división no está definida para los números indicados");
}
else {
    alert("La división es igual a => " + numero1/numero2);
}
```

Infinity, hace referencia a un valor numérico infinito y positivo (también existe el valor **-Infinity** para los infinitos negativos)

```
var numero1 = 10;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor Infinity
```

toFixed(digitos), devuelve el número original con tantos decimales como los indicados por el parámetro **digitos** y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

```
var numero1 = 4564.34567;
numero1.toFixed(2); // 4564.35
numero1.toFixed(6); // 4564.345670
numero1.toFixed(); // 4564
```

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página.

Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.

El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
var enlace = document.getElementById("enlace");  
alert(enlace.href); // muestra http://www...com  
<a id="enlace" href="http://www...com">Enlace</a>
```

En el ejemplo anterior, se obtiene el nodo DOM que representa el enlace mediante la función `document.getElementById()`. A continuación, se obtiene el atributo `href` del enlace mediante `enlace.href`. Para obtener por ejemplo el atributo `id`, se utilizaría `enlace.id`.



Este material forma parte del currículo del módulo **Implantación de Aplicaciones WEB** correspondiente al ciclo formativo de grado superior “**Administración de sistemas informáticos en red**”.

La información contenida en el magazine ha sido extraída de la **Web** y, en parte, de la experiencia profesional acumulada durante mis años de docencia.

Por tanto, es de uso público para tareas de docencia y aprendizaje.