Funciones de Java Script

Las funciones son uno de los bloques de construcción fundamentales en JavaScript. Una función en JavaScript es similar a un procedimiento — un conjunto de instrucciones que realiza una tarea o calcula un valor, pero para que un procedimiento califique como función, debe tomar alguna entrada y devolver una salida donde hay alguna relación obvia entre la entrada y la salida. Para usar una función, debes definirla en algún lugar del ámbito desde el que deseas llamarla.

Programar, usando un símil, podemos verlo como realizar un viaje por carretera. Cuando realizamos un viaje, aparte de la necesidad de definir el objetivo y estudiar la ruta del viaje (estructura del programa) podemos decir que: “En general, pero sobre todo para viajes complicados, conviene dividir el problema en subapartados”.

El concepto de función aplicado a la programación JavaScript es muy similar al aplicable a distintas facetas de la vida: un escritor divide su curso en capítulos y apéndices. Un profesor divide el contenido de la asignatura en temas. Un ingeniero divide el proyecto en partes como Memoria, Anejos, Pliego de Condiciones, Presupuesto y Planos. En una fábrica, organizan el trabajo dividiendo las áreas funcionales en recepción de materias primas, área de pre-proceso, área de proceso, área de post-proceso y área de carga y despacho de producto terminado.

De cara a la programación JavaScript, usaremos la división del código en funciones por ser una estrategia efectiva para resolver problemas complejos. Cada función será llamada para realizar su cometido en un orden establecido.

Además, una función se puede llamar tantas veces como se desee, lo cual evita tener que repetir código y por otro lado permite que cuando haya que realizar una corrección únicamente tengamos que hacerla en la función concreta que se ve afectada.

Las funciones pueden recibir información para realizar su cometido, por ejemplo, function suma (a, b) recibe dos elementos de información: a y b, o no recibirla por

realizar un proceso que no necesita recibir información, por ejemplo function dibujarCirculo().

Otra característica interesante de las funciones es que permite abstraer los problemas. Supongamos que necesitamos una función que devuelva para un importe de una compra sin impuestos el importe con impuestos, y que a su vez el porcentaje de impuestos a aplicar depende del tipo de producto. Si un compañero nos facilita la función function obtenerImporteConImpuestos (importeSinImpuestos) no tenemos que preocuparnos del código de la función. Unicamente sabemos que invocando a la función obtendremos el importe con impuestos. De esta forma, podemos utilizar funciones que han creado otros programadores o funciones disponibles en librerías sin necesidad de conocer el código de las mismas. Decimos que las funciones son "cajas negras" que facilitan la abstracción porque no necesitamos ver en su interior, sólo nos interesan sus resultados.

De hecho, es posible que un programador use un código para una función function obtenerImporteConImpuestos (importeSinImpuestos) y otro programador use otro código para esa misma función sin que esto suponga ningún problema. Lo importante es que la función realice su cometido, no cómo lo realice ya que es frecuente que haya distintas maneras de hacer algo (aunque ciertamente hacer las cosas de diferente manera no debe significar que unas veces se hagan bien y otras mal: siempre deberían hacerse las cosas bien).

Una función en general debe tener un nombre descriptivo de cuál es su cometido y tener un cometido claro y único. No deben mezclarse tareas que no tengan relación entre sí dentro de una función.

Funciones con Parámetros y sin Parámetros

Una función JavaScript puede requerir ser llamada pasándole cierta información o no requerir información.

Definición de una función sin parámetros (no requiere información):

```
// Comentario descriptivo de qué hace la función

function nombreDeLaFunción () {

 // Código de la función

}
```

Definición de una función con parámetros (requiere información):

```
// Comentario descriptivo de qué hace la función

function nombreDeLaFunción (param1, param2, …, paramN) {

 // Código de la función

}
```

Una función puede recibir tantos parámetros como se deseen, aunque no sería demasiado razonable que una función reciba más de cuatro o cinco parámetros. Los parámetros que se le pasan a la función pueden ser:

1. a) Valores simples a los que se denomina literales: por ejemplo 554, true ó ‘aldea’. b) Variables que contienen un número, un texto o un valor booleano. c) Objetos de naturaleza compleja, como arrays y otros tipos de objetos que veremos más

Cuando una función recibe un parámetro dicho parámetro funciona como si se tratara de una variable disponible para la función inicializada con el valor que se le pasa a la función.

Veamos un ejemplo:

```
function mostrarImporteConImpuestos(importeSinImpuestos) {

    var importeConImpuestos; importeConImpuestos = importeSinImpuestos * 1.21;

    msg = ‘Importe antes de impuestos: ‘+ importeSinImpuestos + ‘\n\n’;
```

```
    alert(msg + 'Importe con impuestos: '+ importeConImpuestos + '\n\n');
}
```

Aquí vemos dos cosas de interés: el parámetro que recibe la función no tiene un tipo de datos explícito. El tipo de datos es "inferido" por el intérprete JavaScript. Por otro lado, el parámetro está disponible dentro de la función con el valor con el que haya sido invocado. Por ejemplo, onclick="mostrarImporteConImpuestos(100)" hará que importeSinImpuestos valga 100 porque ese es el valor con el que se invoca. Cuando una función tiene varios parámetros, se debe invocar escribiendo su nombre seguido de los parámetros en el orden adecuado.

Funciones que devuelven un resultado (return)

Una función JavaScript puede devolver un resultado si se introduce la sentencia return resultado; donde resultado es aquello que queremos devolver (normalmente una variable que contiene un valor numérico, de texto o booleano, pero también podrían ser objetos con mayor complejidad como un array).

Una vez se llega a la sentencia return se produce la devolución del resultado y se interrumpe la ejecución de la función. Por ello la sentencia return será normalmente la última instrucción dentro de una función.

Definición de una función sin parámetros que devuelve un resultado: // Comentario descriptivo de qué hace la función

```
function nombreDeLaFunción () {
    // Código de la función
    return resultado;
}
```

Definición de una función con parámetros que devuelve un resultado:

```
// Comentario descriptivo de qué hace la función
function nombreDeLaFunción (param1, param2, …, paramN) {
```

```
    // Código de la función

    return resultado;

}
```

Una función sólo devolverá un resultado y normalmente sólo tendrá una sentencia return, aunque si hay sentencias condicionales como if, puede haber varias sentencias return: una sentencia return para cada sentencia condicional.

Si además del resultado la función incluye código que implique acciones como mostrar un mensaje por pantalla, se ejecutará el código a la vez que se devuelve el resultado.

Veamos un ejemplo:

```
function obtenerImporteConImpuestos(importeSinImpuestos) {

    var importeConImpuestos; importeConImpuestos = importeSinImpuestos * 1.21;

    return importeConImpuestos;

}
```

Un ejemplo de uso de esta función sería:

```
onclick="alert('Calculado para producto de precio 100: importe con impuestos vale ' + obtenerImporteConImpuestos(100));"
```

Llamadas a Funciones desde otras funciones

Una función puede llamar a otra función simplemente escribiendo su nombre y los parámetros que sean necesarios. Ejemplo:

```
function mostrarImporteConImpuestos2(importeSinImpuestos) {

    var msg; msg = 'Ejemplo. Importe antes de impuestos: '+ importeSinImpuestos + '\n\n';

    alert(msg + 'Importe con impuestos: '+

    obtenerImporteConImpuestos(importeSinImpuestos) + '\n\n');

}
```

En esta función en vez de realizarse el cálculo del importe con impuestos, se invoca otra función que es la que se encarga de realizar el cálculo y devolver el valor correspondiente.

Parámetros

Los parámetros son variables locales a los que se les asigna un valor antes de comenzar la ejecución del cuerpo de una función. Su ámbito de validez, por tanto, es el propio cuerpo de la función. El mecanismo de paso de parámetros a las funciones es fundamental para comprender el comportamiento de los programas

Paso de parámetros a Funciones

Hay dos formas comunes de pasar parámetros a funciones en programación: por valor, que implica que si se pasa una variable sus cambios sólo son conocidos dentro de la función, o por variable, que implica que si se pasa una variable ésta puede ser modificada por la función y sus cambios ser conocidos fuera de la función. JavaScript trabaja con paso de parámetros por valor, lo que implica que la variable pasada como parámetro funciona como una variable local a la función: si el parámetro sufre cambios, estos cambios sólo son conocidos dentro de la función. La variable "verdadera" no puede ser modificada.

Paso de un número de parámetros incorrectos

Si se pasan más parámetros de los necesarios, JavaScript ignorará los parámetros sobrantes. Si se pasan menos parámetros de los necesarios, JavaScript asignará valor undefined a los parámetros de los que no se recibe información y se ejecutará sin que surja ningún mensaje de error (aparte de los posibles resultados extraños que esto pudiera ocasionar).

Variables globales y locales

El ámbito de una variable (llamado *"scope"* en inglés) es la zona del programa en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local. El siguiente ejemplo ilustra el comportamiento de los ámbitos:

```
function creaMensaje() {
        var mensaje = “Mensaje de prueba”;
}
creaMensaje();
alert(mensaje);
```

El ejemplo anterior define en primer lugar una función llamada creaMensaje que crea una variable llamada mensaje. A continuación, se ejecuta la función mediante la llamada creaMensaje(); y seguidamente, se muestra mediante la función alert() el valor de una variable llamada mensaje.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable mensaje se ha definido dentro de la función creaMensaje() y por tanto, es una variable local que solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable mensaje.

De esta forma, para mostrar el mensaje en el código anterior, la función alert() debe llamarse desde dentro de la función creaMensaje():

```
function creaMensaje() {
        var mensaje = “Mensaje de prueba”;
        alert(mensaje);
}
creaMensaje();
```

Además de variables locales, también existe el concepto de variable global, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

```
var mensaje = “Mensaje de prueba”;
function muestraMensaje() {
```

```
 alert(mensaje);
}
```

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función muestraMensaje() se quiere hacer uso de una variable llamada mensaje y que no ha sido definida dentro de la propia función.

Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable mensaje.

El motivo es que en el código JavaScript anterior, la variable mensaje se ha definido fuera de cualquier función. Este tipo de variables automáticamente se transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función).

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada mensaje, la variable global creada anteriormente permite que la instrucción alert() dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada var o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante var se consideran locales y las variables que no se han declarado mediante var, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del primer ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada var, para que se transforme en una variable global:

```
function creaMensaje() {
        mensaje = “Mensaje de prueba”;
```

```
}

creaMensaje();

alert(mensaje);
```

¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe? En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

```
var mensaje = "gana la de fuera";

function muestraMensaje() {

        var mensaje = "gana la de dentro";

        alert(mensaje);

}

alert(mensaje);

muestraMensaje();

alert(mensaje);
```

El código anterior muestra por pantalla los siguientes mensajes:

gana la de fuera

gana la de dentro

gana la de fuera

Dentro de la función, la variable local llamada mensaje tiene más prioridad que la variable global del mismo nombre, pero solamente dentro de la función.

¿Qué sucede si dentro de una función se define una variable global con el mismo nombre que otra variable global que ya existe? En este otro caso, la variable global definida dentro de la función simplemente modifica el valor de la variable global definida anteriormente:

```
var mensaje = "gana la de fuera";
```

```
function muestraMensaje() {
        mensaje = “gana la de dentro”;
        alert(mensaje);
}
alert(mensaje);
muestraMensaje();
alert(mensaje);
```

En este caso, los mensajes mostrados son:

gana la de fuera

gana la de dentro

gana la de dentro