El futuro digital es de todos

MinTIC

UNIVERSIDAD EL BOSQUE

Mision TIC 2022

Recurso 2.1

JavaScript Funciones

Mag. Carlos Adolfo Beltrán Castro

Operadores

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.

Funciones

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;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
numero1 = 5;
numero2 = 8;

// Se suman los números y se muestra el resultado
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 */
...
```

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 { y } 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.

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.

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);
```

Ambito de las variables

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 muestra por pantalla los siguientes mensajes:

```
gana la de fuera
gana la de dentro
gana la de fuera
```

Estructura Switch

La estructura if...else se puede utilizar para realizar comprobaciones múltiples y tomar decisiones complejas. Sin embargo, si todas las condiciones dependen siempre de la misma variable, el código JavaScript resultante es demasiado redundante:

```
if(numero == 5) {
   ...
}
else if(numero == 8) {
   ...
}
else if(numero == 20) {
   ...
}
else {
   ...
}
```

En estos casos, la estructura switch es la más eficiente, ya que está especialmente diseñada para manejar de forma sencilla múltiples condiciones sobre la misma variable. Su definición formal puede parecer compleja, aunque su uso es muy sencillo:

```
switch(variable) {
  case valor_1:
    ...
    break;
  case valor_2:
    ...
    break;
  ...
  case valor_n:
    ...
    break;
  default:
    ...
    break;
}
```

La estructura switch se define mediante la palabra reservada switch seguida, entre paréntesis, del nombre de la variable que se va a utilizar en las comparaciones. Como es habitual, las instrucciones que forman parte del switch se encierran entre las llaves { y }.

Dentro del switch se definen todas las comparaciones que se quieren realizar sobre el valor de la variable. Cada comparación se indica mediante la palabra reservada case seguida del valor con el que se realiza la comparación. Si el valor de la variable utilizada por switch coincide con el valor indicado por case, se ejecutan las instrucciones definidas dentro de ese case.

Normalmente, después de las instrucciones de cada case se incluye la sentencia break para terminar la ejecución del switch, aunque no es obligatorio. Las comparaciones se realizan por orden, desde el primer case hasta el último, por lo que es muy importante el orden en el que se definen los case.

¿Qué sucede si ningún valor de la variable del switch coincide con los valores definidos en los case? En este caso, se utiliza el valor default para indicar las instrucciones que se ejecutan en el caso en el que ningún case se cumpla para la variable indicada.