APRENDERAPROGRAMAR.COM

CLOSURES JAVASCRIPT. EJEMPLOS. CONCEPTO: QUÉ SON Y PARA QUÉ SIRVEN. RETARDO DE EJECUCIÓN CON SETTIMEOUT Y CLOSURES. (CU01169E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº69 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CLOSURES JAVASCRIPT

Los closures JavaScript son funciones que llevan información asociada relativa al momento en que son invocadas. No es fácil explicar ni entender el concepto de closure porque éste es un tanto complejo. Recomendamos que se estudie viendo ejemplos y se vaya asimilando poco a poco a medida que se practique con el desarrollo de código JavaScript.

Los closures, en castellano denominados cierres, cerraduras o clausuras, son una característica de algunos lenguajes entre los que se encuentra JavaScript.

Un closure se genera cuando se produce la siguiente situación en el código:

```
    function funcionExterna(par1, par2, ..., parN){
        var miVariableLocal = un valor;
        var miFuncionInterna = function () {
                return par1 - miVariableLocal; // Situación que genera el closure
        }
        return miFuncionInterna ó miFuncionInterna(); // Veremos ejemplos para entenderlo
}
```

El closure se genera cuando una función interna a otra función usa una variable local (o parámetro recibido) de la función externa. Con un ejemplo lo veremos más claro:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function dimeMsg(nombre){
        var msg = 'hola';
        var respuesta = function () { alert(msg+ ' ' + nombre); }
        respuesta()
}

function ejemplo(){ var habla1 = dimeMsg('Juan'); }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla <<Hola Juan>>

La función interna también la podemos definir así (siendo equivalente al código anterior):

```
function dimeMsg(nombre){
        var msg = 'hola';
        function respuesta () {  alert(msg+ ' ' + nombre); }
        respuesta()
}
```

A las funciones internas que hacen uso de variables locales de las funciones externas dentro de las cuales se encuentran las denominamos cerraduras o closures. Una cerradura tiene unas particularidades que trataremos de estudiar a continuación. Ten en cuenta que los closures a veces se generan "intencionadamente" y otras veces se generan "sin querer". Pero de una forma u otra, conviene entender qué implica que exista un closure para poder entender lo que ocurre en muchos scripts.

El código anterior parece que tiene poco interés, pero veamos cómo los closures tienen características interesantes.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function dimeVuelta(entrega){
        var precio = 1000;
        var respuesta = function () { return precio-entrega; } //Aquí el closure
        return respuesta;
}

function ejemplo(){
        var calcula = dimeVuelta(600);
        alert('Su vuelta es ' +calcula());
}

</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla: <<Su vuelta es: 400 >>

Analicemos el código. La referencia "respuesta" es una cerradura o closure, ya que es una función interna que utiliza variables locales de la función externa.

La función externa "dimeVuelta" devuelve como resultado una referencia a la función interna.

Al ejecutar var calcula = dimeVuelta(600); lo que se almacena en "calcula" es una referencia a la función interna. Es como si hiciéramos calcula = function () { return precio-entrega; } ¿Pero qué significado tendrían aquí precio y entrega, teniendo en cuenta que la función donde se definían ya se ejecutó y por tanto en principio están fuera de un ámbito válido? Teóricamente las variables globales a una función son destruidas cuando termina de ejecutarse la función, por tanto precio y entrega supuestamente deberían haber sido destruidas.

Sin embargo, cuando JavaScript encuentra un closure toda variable local que sea necesaria para el funcionamiento del closure queda encerrada en el propio closure. Es decir, dado que "respuesta" necesita de "precio" y "entrega", éstas se guardan dentro de la función cerradura.

Después de ejecutarse var calcula = dimeVuelta(600); en la variable "calcula" tenemos almacenados el precio (1000) y la entrega (600), aunque la función externa ya haya sido ejecutada.

Ahora calcula tiene una referencia a una función. Para ejecutar dicha función invocamos calcula(), y dado que esta función recuerda los valores de variables locales devuelve 400 (obtenidos de 1000-400, precio-entrega).

Ahora bien, ¿qué valor de variable local es el que almacena el closure? Tener en cuenta que una variable local puede cambiar a lo largo del código. Por ejemplo:

```
function dimeVuelta(entrega){
        var precio = 1000;
        var respuesta = function () { return precio-entrega; } //Aquí el closure
        precio = 700;
        return respuesta;
}
```

¿El closure quedará tomando como referencia 1000 ó 700? La realidad es que toma como referencia el valor que tenía la variable local cuando se produce la salida de la función externa (en este ejemplo justo antes del return), por tanto en este caso el closure queda almacenando como precio un valor de 700.

Podría darse la situación de que existan varias funciones internas a una función externa dada, y que varias de esas funciones internas usen variables locales de la función externa. En este caso decimos que se generan varias cerraduras (una por cada función interna que hace uso de variables locales), pero aquí sí es cierto que todas ellas quedan con una única referencia de variable local: la que exista cuando se produzca la salida de la función externa.

En el ejemplo anterior hemos usado la creación de una referencia intermedia para después invocar la función:

```
function ejemplo() {        var calcula = dimeVuelta(600);        alert('Su vuelta es ' +calcula()); }
```

Pero la invocación de la función podemos hacerla directamente si lo deseamos escribiendo esto:

```
function ejemplo() {  alert('Su vuelta es ' +dimeVuelta(600)()); }
```

Aquí dimeVuelta(600) nos devuelve la referencia a la función anónima, y al añadir () a continuación, damos pie a su ejecución directamente (sin necesidad de crear la referencia usando var).

Si escribes alert('Su vuelta es ' +dimeVuelta(600)); por pantalla obtendrás la función que devuelve la función invocada, por tanto se verá << Su vuelta es function () { return precio-entrega; }>> o un mensaje similar (puede variar según el navegador).

CADA CLOSURE LLEVA SUS DATOS

Cuando se invoca la función externa se genera un closure y cada closure que se genere guarda sus propias referencias, es decir, no se guarda una única referencia para todos los closures. Un closure es una combinación de función y de datos relativos al momento de su creación. En ese sentido podemos decir que recuerdan a los objetos de la programación orientada a objetos (objeto = datos + métodos closure = datos + función). Un closure sería un objeto con un solo método. Ejecuta este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function dimeVuelta(entrega){            var precio = 1000;
            var respuesta = function () { return precio-entrega; } //Aquí el closure
            return respuesta;
}

function ejemplo(){ var calculo1 = dimeVuelta(600);          alert('Su vuelta es: '+calculo1());
                        var calculo2 = dimeVuelta(500);          alert('Su vuelta es: '+calculo2());
}

</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

El resultado es <<Su vuelta es: 400>> y <<Su vuelta es: 500>> porque cada closure guarda la referencia al valor de las variables locales en el momento en que fueron creados.

El hecho de que cada closure guarde su información permite interesantes aplicaciones.

USAR CLOSURES PARA FUNCIONES RETARDADAS

Partimos del siguiente código con el que tratamos de hacer que mediante un bucle for se cuente de 1 a 10 con intervalos de 1 segundo entre que aparezca cada número en pantalla. Ejecuta el código y comprueba qué ocurre:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
*{font-family: verdana, sans-serif;}
.nodoNuevo{background-color: black; color:white; width:100px;
text-align:center; padding:20px; font-size:32px; float:left;}
</style>
<script type="text/javascript">

function mostrarNumConRetardo() {
for(var i=1; i<11; i++){ setTimeout(crearNodo(i), 1000); }
}

function crearNodo(numero) {
var nodoHijo = document.createElement("div");
nodoHijo.className="nodoNuevo"; nodoHijo.innerHTML = ''+numero;
document.body.appendChild(nodoHijo);
}
</script>
</script></head>
<body onload="mostrarNumConRetardo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

Lo que ocurre es que se muestra por pantalla de forma inmediata todos los números (1-2-3-4-5-6-7-8-9-10) sin retardo.

Podemos pensar en tratar de arreglarlo con la siguiente modificación:

```
function mostrarNumConRetardo() {
for(var i=1; i<11; i++){ setTimeout(crearNodo(i), 1000*i); }
}
```

Pero esto no funciona. ¿Por qué? Porque el valor de i que se está pasando a setTimeout no es el valor de i en cada bucle, sino la referencia a la variable i cuando setTimeout se ha ejecutado y el bucle ha terminado, y esa referencia no tiene valor (ya que el bucle ha terminado).

Necesitamos que setTimeout “recuerde” el valor que tenía i en cada pasada del bucle. Esto lo podemos hacer creando un closure en cada pasada del bucle. Ejecuta este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
*{font-family: verdana, sans-serif;}
.nodoNuevo{background-color: black; color:white; width:100px;
text-align:center; padding:20px; font-size:32px; float:left;}
</style>
<script type="text/javascript">
```

```
function mostrarNumConRetardo() {
for(var i=1; i<11; i++){ setTimeout(function(x) { return function() { crearNodo(x); }; }(i), 1000*i); }
}

function crearNodo(numero) {
var nodoHijo = document.createElement("div");
nodoHijo.className="nodoNuevo"; nodoHijo.innerHTML = ''+numero;
document.body.appendChild(nodoHijo);
}
</script>
</script></head>
<body onload="mostrarNumConRetardo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

Dentro del setTimeout invocamos la siguiente función:

function(x) { return function() { crearNodo(x); }; }(i)

Con esta sintaxis lo que hacemos es crear una función anónima que recibe un parámetro x y que devuelve otra función anónima interna que crea un closure ya que usa el parámetro que recibe la función externa. El closure recuerda el valor de la variable local (parámetro). Para ejecutar el closure, invocamos la función externa pasándole (i) como parámetro, siendo i el contador del bucle. Esto fuerza que setTimeout se ejecute con los valores que tenía i en cada pasada del bucle, y no con una única referencia a i.

RESUMEN SOBRE CLOSURES Y ÁMBITOS

Los closures son funciones que llevan datos asociados, relativos al momento en que fueron invocadas.

La existencia de closures aporta ventajas a la programación con JavaScript, ya que podemos usarlos para resolver necesidades que nos surjan. Pero también genera problemas: a veces se generan closures sin querer con efectos indeseados. O a veces se crea un excesivo número de closures innecesariamente, consumiendo recursos y haciendo más lenta la ejecución del código.

Los closures son una parte de la programación JavaScript que no es fácil de explicar ni de entender. Esto podemos extenderlo en general a “los ámbitos” y a la palabra clave this. No te preocupes si te has perdido en algunas partes de las explicaciones que hemos dado. Sigue avanzando con el curso y trata de ir adquiriendo destreza en la interpretación y uso de closures a medida que sigas programando JavaScript.

EJERCICIO

Analiza el siguiente código y responde a las siguientes preguntas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">

<style type="text/css">
body { font-family: Helvetica, Arial, sans-serif;}
h2 {  font-size: 1.5em;} h3 { font-size: 1.2em;}
div div {color:blue; margin:10px;}
</style>

<script type="text/javascript">
function cambiarDimensionFuente(size) {   return function() { document.body.style.fontSize = size + 'px';};
}

var size8 = cambiarDimensionFuente(8);
var size16 = cambiarDimensionFuente(16);
var size24 = cambiarDimensionFuente(24);

function setClicks(){
document.getElementById('fuente-8').onclick = size8;
document.getElementById('fuente-16').onclick = size16;
document.getElementById('fuente-24').onclick = size24;
}

</script></head>
<body onload="setClicks()">
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<div id ="fuente-8" > Poner texto a 8 </div> <div id ="fuente-16" > Poner texto a 16 </div>
<div id ="fuente-24" > Poner texto a 24 </div> </div>
<p>En las praderas de la estepa de la Tierra del Fuego suele hacer frío</p>
</body></html>
```

a) ¿En qué parte del código se genera un closure o cerradura? ¿Por qué?

b) ¿En qué parte del código se establece que al hacer click sobre el elemento con id fuente-8 se cambie el tamaño de las fuentes de la página?

c) Supón que eliminamos la función setClicks y dejamos su código “libre” dentro de las etiquetas <script> … </script>. ¿Qué mensaje de error te muestra la consola de depuración? (Activa la consola si no la tienes activada) ¿Por qué aparece ese mensaje de error?

d) ¿Debemos escribir document.getElementById('fuente-8').onclick = size8; ó document.getElementById('fuente-8').onclick = size8(); ?¿Por qué?

e) Supón que al cargar la página queremos que el tamaño inicial de fuente sea 8 y para ello nos valemos de la función setClicks. ¿Debemos escribir dentro de esta función size8; ó size8();? ¿Por qué?

f) Las closures no siempre son necesarias, incluso a veces se generan involuntariamente o innecesariamente consumiendo recursos del sistema que podrían ahorrarse. ¿Qué ventajas le ves al uso de closures en este código? ¿Y qué inconvenientes?

g) Reescribe el código (hazlo como mejor creas cambiando todo aquello que consideres necesario) de forma que obtengamos el mismo resultado pero sin hacer uso de closures.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01170E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206