

MongoDB Fundamentos

Módulo 3

JavaScript en MongoDB

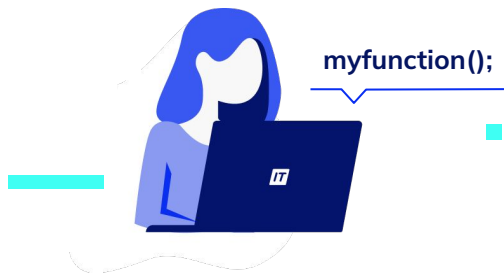
Creación de funciones

Una de las tareas más importantes del desarrollo con JavaScript es **crear código que otro código pueda reutilizar**. Para hacer esto, se debe organizar el código en funciones que realicen tareas específicas. Una función es **una serie de instrucciones combinadas en un solo bloque al que se le asigna un nombre**. El código en el bloque se puede ejecutar haciendo referencia a ese nombre.



Definición de funciones

Las funciones se definen mediante la palabra clave **function** seguida de un nombre que describe el uso de la función, una lista de cero o más argumentos entre paréntesis y un bloque de una o más declaraciones de código entre `{}` corchetes.



Por ejemplo, la siguiente es una definición de función que escribe "Hola Mundo" en la consola.

```
function myFunction(){  
  print("Hola Mundo");  
}
```

Para ejecutar el código incluido en `myFunction()`, todo lo que se necesita hacer es agregar la siguiente línea al JavaScript principal o dentro de otra función:

```
myFunction();
```

Pasar variables a funciones

Con frecuencia, se deben pasar valores específicos a las funciones que se utilizarán. Los valores se pasan delimitados por comas. La definición de la función necesita una lista de nombres de variables entre paréntesis `()` que coincidan con el número que se pasa. Por ejemplo, esta función acepta dos argumentos, **nombre** y **ciudad**, y los usa para construir la cadena de salida:

```
function saludo(nombre, ciudad){  
  print("Hola " + nombre);  
  print("Cómo está el clima en " + ciudad);  
}
```

Para llamar a la función `saludo()`, se debe pasar el valor **nombre** y un valor para **ciudad**. El valor puede ser directo o una variable previamente definida. Para ilustrar esto, el siguiente código ejecuta la función `saludo()` con una variable **nombre** y una cadena directa para **ciudad**:

```
var nombre = "Pablo";  
greeting(nombre, "Florenia");
```



Devolver valores de funciones

Con frecuencia, las funciones necesitan devolver un valor al código de llamada. Agregar la palabra clave **return** seguida de una variable o valor devuelve ese valor de la función.

Por ejemplo, el código de la derecha llama a una función para formatear una cadena, asigna el valor devuelto por la función a una variable y luego escribe el valor en la consola:

```
function formatoSaludo(nombre, ciudad){  
  var retStr = "";  
  retStr += "Hola " + nombre + "\n";  
  retStr += "Bienvenido a " + ciudad + "!";  
  return retStr;  
}  
var saludo = formatoSaludo("Carlos", "Roma");  
print(saludo);
```

Se puede incluir más de una instrucción **return** en la función. Cuando la función encuentra una declaración **return**, la ejecución del código de la función se detiene inmediatamente. Si la declaración de devolución contiene un valor para devolver, se devuelve ese valor.

El siguiente ejemplo muestra una función que prueba la entrada y regresa inmediatamente si es cero.

```
function miFunc(valor){  
  if (valor == 0)  
    return valor;  
  <código a ejecutar si el valor es distinto de cero>  
  return valor;  
}
```





Manipulación de cadenas

El objeto **String** es el más utilizado en JavaScript. **JavaScript crea automáticamente un objeto String cada vez que se define una variable del tipo de datos de cadena.** Por ejemplo:

```
var myStr = "Aprende NoSQL con MongoDB en 24 horas";
```

Al crear una cadena, varios caracteres especiales no se pueden agregar directamente. Para estos caracteres, JavaScript proporciona un conjunto de códigos de escape, veamos la tabla de la siguiente slide.



| Escape | Descripción | Ejemplo | Cadena de salida |
|--------|------------------|-----------------------------|----------------------|
| \' | Comilla simple | "couldn\'t be" | couldn't be |
| \" | Comillas dobles | "Yo \ "pienso\" Yo \"soy\"" | Yo “pienso” Yo “soy” |
| \\ | Barra invertida | "uno\\dos\\tres" | uno\dos\tres |
| \n | Nueva línea | "Yo soy\nYo dije" | Yo soy Yo dije |
| \r | Retorno de carro | "ser\rro no ser" | Ser o no ser |
| \t | Tabulación | "uno\tdos\ttres" | uno dos tres |
| \b | Backspace | "correctoin\b\b\bion" | correction |
| \f | Form feed | "Title A\fTitle B" | Title A then Title B |

Para obtener la longitud de la cadena, se puede usar la propiedad **length** del objeto **String**.

Por ejemplo:

```
var numOfChars = myStr.length;
```


El objeto **String** tiene varias funciones que le permiten acceder y manipular la cadena de varias maneras. La tabla que se muestra a continuación describe los métodos de manipulación de cadenas.



| Método | Description |
|--|---|
| <code>concat (arr1, arr2, ...)</code> | Devuelve una copia concatenada del array y los arrays pasados como argumentos. |
| <code>indexOf(value)</code> | Devuelve el índice de un valor del array o -1 si no se encuentra el elemento. |
| <code>join(separator)</code> | Une todos los elementos de un array, que están separados por el separador indicado, en una sola cadena. Si no se especifica ningún separador, se utiliza una coma. |
| <code>lastIndexOf(value)</code> | Devuelve el último índice del valor en el array o -1 si no se encuentra el valor. |
| <code>pop()</code> | Elimina el último elemento de la matriz y devuelve ese elemento. |
| <code>push(item1, item2, ...)</code> | Agrega uno o más elementos nuevos al final de una matriz y devuelve la nueva longitud. |
| <code>reverse()</code> | Invierte el orden de todos los elementos del array. |
| <code>shift()</code> | Elimina el primer elemento de un array y devuelve ese elemento. |
| <code>slice(start, end)</code> | Devuelve los elementos entre el índice inicial y final. |
| <code>sort(sortFunction)</code> | Ordena los elementos del array. La función <code>sortFunction</code> es opcional. |
| <code>splice(index, count, item1, item2...)</code> | En el índice especificado, se elimina el número de elementos que indica <code>count</code> . Luego, los elementos opcionales que se pasan como argumentos se insertan en el índice. |
| <code>toString()</code> | Devuelve el array como cadena. |
| <code>unshift()</code> | Agrega nuevos elementos al comienzo de un array y devuelve la nueva longitud. |
| <code>valueOf()</code> | Devuelve el valor primitivo de un arreglo de objetos. |

Combinación de cadenas

Se pueden combinar varias cadenas usando el signo `+` o la función **concat()** en la primera cadena. Por ejemplo, en el siguiente código, `sentencia1` y `sentencia2` harán lo mismo:

```
var palabra1 = "Hoy ";  
var palabra2 = "es ";  
var palabra3 = "mañana\' ";  
var palabra4 = "ayer.";   
var sentencia1 = palabra1 + palabra2 + palabra3 + palabra4;  
var sentencia2 = palabra1.concat(palabra2, palabra3, palabra4);
```

Trabajar con matrices

El objeto **array** proporciona un medio para almacenar y manejar un conjunto de otros objetos. Las matrices pueden almacenar números, cadenas u otros objetos de JavaScript. Existen varios métodos para crear matrices de JavaScript.

Por ejemplo, las siguientes declaraciones crean tres versiones idénticas de la misma matriz:

```
var arr = ["uno", "dos", "tres"];
```

```
var arr2 = new Array();  
arr2[0] = "uno";  
arr2[1] = "dos";  
arr3[2] = "tres";
```

```
var arr3 = new Array();  
arr3.push("uno");  
arr3.push("dos");  
arr3.push("tres");
```



El primer método define **arr** y establece el contenido en una sola declaración usando `[]`.

El segundo método crea el objeto **arr2** y luego le agrega elementos mediante la asignación de índice directo.

El tercer método crea el objeto **arr3** y luego usa la mejor opción para extender matrices: usa el método **push()** para insertar elementos en la matriz.

Para obtener la cantidad de elementos en la matriz, usa la propiedad `length` del objeto Array. Veamos un ejemplo:

```
var numDeItems = arr.length;
```

Las matrices son un índice basado en cero, lo que significa que el primer elemento está en el índice 0, y así sucesivamente. En este ejemplo, el valor de la variable **primero** es **lunes** y el valor de la variable **ultimo** es **viernes**:

```
var semana = ["lunes", "martes",  
              "miércoles", "jueves", "viernes"];  
var primero = w [0];  
var ultimo = semana[semana.length-1];
```

El objeto Array tiene varias funciones integradas que le permiten acceder y manipular la matriz de varias maneras. La siguiente tabla describe los métodos adjuntos al objeto Array que le permiten manipular el contenido de la matriz.

| Método | Description |
|--|---|
| <code>concat (arr1, arr2, ...)</code> | Devuelve una copia concatenada del array y los arrays pasados como argumentos. |
| <code>indexOf(value)</code> | Devuelve el índice de un valor del array o -1 si no se encuentra el elemento. |
| <code>join(separator)</code> | Une todos los elementos de un array, que están separados por el separador indicado, en una sola cadena. Si no se especifica ningún separador, se utiliza una coma. |
| <code>lastIndexOf(value)</code> | Devuelve el último índice del valor en el array o -1 si no se encuentra el valor. |
| <code>pop()</code> | Elimina el último elemento de la matriz y devuelve ese elemento. |
| <code>push(item1, item2, ...)</code> | Agrega uno o más elementos nuevos al final de una matriz y devuelve la nueva longitud. |
| <code>reverse()</code> | Invierte el orden de todos los elementos del array. |
| <code>shift()</code> | Elimina el primer elemento de un array y devuelve ese elemento. |
| <code>slice(start, end)</code> | Devuelve los elementos entre el índice inicial y final. |
| <code>sort(sortFunction)</code> | Ordena los elementos del array. La función <code>sortFunction</code> es opcional. |
| <code>splice(index, count, item1, item2...)</code> | En el índice especificado, se elimina el número de elementos que indica <code>count</code> . Luego, los elementos opcionales que se pasan como argumentos se insertan en el índice. |
| <code>toString()</code> | Devuelve el array como cadena. |
| <code>unshift()</code> | Agrega nuevos elementos al comienzo de un array y devuelve la nueva longitud. |
| <code>valueOf()</code> | Devuelve el valor primitivo de un arreglo de objetos. |

Combinación de matrices

Es posible combinar matrices de la misma manera que se combinan objetos `String`, usando el signo `+`, instrucciones o el método `concat()`.

En el siguiente código, `arr3` hace lo mismo que `arr4`:

```
var arr1 = [1,2,3];  
var arr2 = ["tres", "cuatro", "cinco"]  
var arr3 = arr1 + arr2;  
var arr4 = arr1.concat(arr2);
```



Iterar a través de matrices

Es posible iterar a través de una matriz usando **for** o **for/ inloop**. El siguiente código muestra la iteración a través de cada elemento de la matriz utilizando cada método:

```
var semana = ["lunes", "martes", "miércoles", "jueves", "viernes"];  
for (var i=0; i<semana.length; i++){  
    print(semana[i] + "\n");  
}  
for (dayIndex in semana){  
    print(semana[dayIndex] + "\n");  
}
```



Uso de objetos de JavaScript

JavaScript tiene varios objetos integrados, como `Number`, `Array`, `String`, `Date` y `Math`. Cada objeto incorporado tiene propiedades y métodos de miembros. Además de los objetos de JavaScript, el shell de MongoDB proporciona objetos integrados.

JavaScript proporciona una estructura de programación orientada a objetos. El uso de objetos en lugar de solo una colección de funciones es clave para escribir código JavaScript limpio, eficiente y reutilizable.



Sintaxis de objetos

Para usar objetos en JavaScript de manera efectiva, se debe comprender su estructura y sintaxis. Un objeto es realmente solo un contenedor para agrupar múltiples valores y, en algunos casos, funciones. Los valores de un objeto se llaman **propiedades** y las funciones se denominan **métodos**.

Para usar un objeto de JavaScript, primero se debe crear una instancia de ese objeto. Las instancias de objetos se crean utilizando la palabra clave **new** con el nombre del constructor del objeto.

Por ejemplo, para crear un objeto `Numero`, se utiliza la siguiente línea de código:

```
var x = new Numero("5");
```

La sintaxis de objetos es sencilla: el nombre del objeto, luego un punto y luego el nombre de la propiedad o del método. Por ejemplo, estas líneas de código obtienen y establecen la propiedad **name** de un objeto denominado `myObj`:

```
var s = myObj.name;  
myObj.name = "Nuevo nombre";
```

También puede obtener y establecer métodos de un objeto de la misma manera. Por ejemplo, el código de la derecha llama al `getName()` método y luego cambian la función del método en un objeto llamado `myObj`.

```
var name = myObj.getName();  
myObj.getName = function() { return this.name;  
};
```

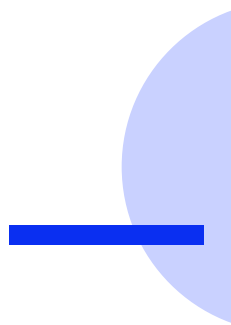
También puede crear objetos en forma literal y asignar variables y funciones directamente usando la sintaxis `{}`. Por ejemplo, el código a la derecha, define un nuevo objeto y asigna valores y una función de método.

```
var obj = {  
  name: "My Object",  
  value: 7,  
  getValue: function() { return this.value; }  
};
```

También se puede acceder a los miembros de un objeto de JavaScript utilizando la sintaxis `object[propertyName]`. Esto es útil cuando se utilizan nombres de propiedades dinámicas o si el nombre de la propiedad debe incluir caracteres que JavaScript no admite.

Los siguientes ejemplos acceden a las propiedades "Nombre de usuario" y "Otro nombre" de un objeto llamado `myObj`:

```
var propName = "Nombre de usuario";  
var val1 = myObj[propName];  
var val2 = myObj["Otro nombre"];
```



Creación de objetos definidos personalizados

Los objetos integrados de JavaScript tienen varias ventajas. A medida que se comience a escribir código que utilice más datos, se querrá crear **objetos personalizados** con propiedades y métodos específicos.

El objeto de JavaScript se puede definir de un par de maneras diferentes. El más simple es el **método sobre la marcha**, lo que significa que se crea un objeto genérico y luego se le agregan propiedades a medida que se necesitan.

Por ejemplo, el siguiente código crea un objeto de usuario, le asigna un nombre y apellido y define una función para devolver el nombre completo:

```
var user = new Object();  
user.first="Carlos";  
user.last="Pérez";  
user.getName = function( ) { return  
this.first + " " + this.last; }
```



Se lograría el mismo efecto a través de una **asignación directa** usando la siguiente sintaxis.

El objeto está entre `{}` corchetes y las propiedades se definen mediante `property:value`:

```
var user = {  
  first: 'Carlos',  
  last: 'Pérez',  
  getName: function( ) { return this.first + " " + this.last; }  
};
```

Estas dos primeras opciones funcionan bien para objetos simples que no se necesitan reutilizar más adelante.

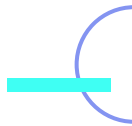
Un mejor método para los objetos reutilizables es **encerrar el objeto dentro de su propio bloque de funciones**. Tiene la ventaja de permitirle mantener todo el código perteneciente al objeto local para el propio objeto.

Veamos:

```
function User(first, last){  
  this.first = first;  
  this.last = last;  
  this.getName = function( ) { return this.first + " " + this.last; }  
};  
var user = new User("Carlos", "Pérez");
```

El resultado final de estos métodos es esencialmente el mismo. Se tiene un objeto con propiedades a las que se puede hacer referencia mediante la sintaxis de puntos:

```
print(user.getName());
```



Fuente

Sams Teach Yourself NoSQL with MongoDB in 24 Hours - Brad Dayley - Pearsen Education 2015



**¡Sigamos
trabajando!**