Funciones en JavaScript

Las Funciones permiten agrupar líneas de código que pueden ser reutilizadas mediante un nombre o una referencia a él. Por lo general, aceptan varios parámetros de entrada y generan valores de salida (**sólo uno por función**).

Declaración de la función

```
function sum( a, b ) {
    var c = a + b;
    return c;
}
```

Parámetros

Los parámetros de una función son como variables locales que reciben el valor asignado por la función que invocó. Ellos son opcionales, por definición. Las funciones pueden usarse con un número indeterminado de ellos.

Ejemplo:

```
function paramTest( a, b ) {
  console.log( a, b );
}
paramTest();
//> undefined undefined

paramTest( 'hello' );
//> hello undefined

paramTest( 'hello', 'world' );
//> hello world

paramTest( 'hello', 'world', '!' );
//> hello world
```

La última llamada no falla, porque **podemos llamar a una función con más argumentos de los que espera**.

Podemos acceder a todos los argumentos usados en la llamada de una función usando `arguments` dentro de esta función:

```
function argumentsTest( a, b ) {
   console.log( arguments );
}

argumentsTest();
//> []

argumentsTest( 'hello' );
//> [ 'hello' ]

argumentsTest( 'hello', 'world' );
//> [ 'hello', 'world' ]

argumentsTest( 'hello', 'world', '!' );
//> [ 'hello', 'world', '!' ]
```

`arguments` no es un array, aunque se vea como un array. Es un objeto que contiene keys desde `0` a `arguments.length` (cantidad de argumentos).

Los parámetros en JavaScript se pasan por referencia, por lo tanto un cambio en el objeto pasado como parámetro, cambiará el objeto original:

```
function renameAsMartin(object) {
   object.name = 'Martin';
}
var obj = { name: 'John', surname: 'Doe' };

// obj {name: "John", surname: "Doe"}

renameAsMartin( obj );
obj.name === 'Martin';

// obj {name: "Martin", surname: "Doe"}
```

Funciones Globales o pre-definidas

Hay una serie de funciones que están directamente definidas dentro del motor de JavaScript. En este documento vamos a ver algunas de estas funciones:

- parseInt()
- parseFloat()
- isNaN()
- isFinite()
- encodeURI()

- decodeURI()
- encodeURIComponent() o decodeURIComponent() o eval()

parseInt()

A partir de un valor intenta transformarlo en número entero. Si falla devuelve NaN.

parseInt() admite un segundo parámetro opcional que indica la base del numero que se le está pasando (decimal, hexadecimal, binario, etc.)

https://developer.mozilla.org/es/Referencia_de_JavaScript_1.5/Funciones_globales/parseInt

Ejemplo:

```
>>> parseInt('123')
123
>>> parseInt('abc123')
NaN
>>> parseInt('1abc23')
1
>>> parseInt('123abc')
123
```

parseFloat()

A partir de un valor intenta transformarlo en número de coma flotante (con decimales).

Ejemplo:

```
>>> parseFloat('123')
123
>>> parseFloat('1.23')
1.23
>>> parseFloat('1.23abc.00')
1.23
>>> parseFloat('a.bc1.23')
NaN
```

isNan()

Comprueba si el valor que se le pasa es un número válido (devuelve `true` en caso de que no lo sea)

Ejemplo:

```
>>> isNaN(NaN)
true
>>> isNaN(123)
false
>>> isNaN(1.23)
false
>>> isNaN(parseInt('abc123'))
true
```

isFinite()

Comprueba si el valor que se le pasa no es ni `Infinity` ni `NaN`

Ejemplo:

```
>>> isFinite(Infinity)
false
>>> isFinite(-Infinity)
false
>>> isFinite(12)
true
>>> isFinite(1e308)
true
>>> isFinite(1e309)
false
```

encodeURI()

Nos permite 'escapar' (codificar) una URL reemplazando algunos caracteres por su correspondiente secuencia de escape UTF-8. encodeURI()nos devuelve una URL usable (solo codifica algunos caracteres)

Ejemplo:

```
>>> var url = 'http://www.senderators.com/scr ipt.php?q=esto es un
texto';
>>> encodeURI(url);
http://www.senderators.com/scr%20ipt.php?q=esto%20es%20un%20texto
```

decodeURI()

Nos permite 'decodificar' un string codificado por encodeURI()

encodeURIComponent() y decodeURIComponent()

Lo mismo que encodeURI() pero esta función codifica (decodifica) TODOS los caracteres transformables.

Ejemplo:

```
>>> encodeURIComponent(url);
"http%3A%2F%2Fwww.senderators.com%2Fscr%20ipt.php%3Fq%3Desto%20es%20un%20texto"
```

eval()

Toma una cadena de texto y la ejecuta como código JavaScript

eval() no debe utilizarse básicamente por 2 motivos:

- **Rendimiento**: Es mucho más lento evaluar código "en vivo" que tenerlo directamente en el script
- **Seguridad**: Teniendo en cuenta que ejecuta cualquier código que se le pase puede ser un agujero de seguridad importante.

Ejemplo:

```
>>> eval('var ii = 2;')
>>> ii 2
```

alert()

Nos muestra una ventana con un string

alert() no es parte del core JS pero está disponible en todos los navegadores. **¡Cuidado!** alert() para el código hasta que se acepte el mensaje

Mejor usar console.log para el debug.

Lee la documentación en Mozilla Developer Center sobre los Objetos y Funciones Globales de JavaScript https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales y amplia información. Ten presente esta documentación para desarrollar en JavaScript.

Funciones como first-class objects

Las funciones son como otras variables en JavaScript. Podemos declarar funciones dentro de otras funciones y utilizarlas como usaríamos las variables: el acceso a sus propiedades, utilizándolas como parámetros. Estos parámetros en JavaScript se denominan callbacks.

```
function myFunction( a, b ) {
   console.log( a, b );
}
var functionWithOtherName = myFunction;
myFunction( 'hello', 'world' );
//> hello world
functionWithOtherName( 'hello', 'world' );
//> hello world
myFunction === functionWithOtherName
// true
```

Callbacks

Las funciones en JavaScript son datos, lo que significa que podemos asignarlas a variables igual que cualquier otro valor (y manejarlas como variables).
Si pasamos una función como parámetro de otra función, está puede ser utilizada como callback.

Veamos un ejemplo:

```
function showResult( result ) {
   alert( result );
}
function addAndCallWithResult( val1, val2, callback ) {
   var sum = val1 + val2;
   callback( sum );
}
addAndCallWithResult( 1, 5, showResult );
//> alert 6!
function sayHello() {
   alert( 'Hello!' );
}
```

```
setTimeout(sayHello, 1000); /
/> after 1s: alert 'Hello!'
```

Las funciones sayHello() y showResult(), se ejecutan dentro de otra función como callbacks.

Funciones como expresiones (Function expressions)

Hasta ahora, hemos definido las funciones usando una declaración:

```
function functionName( param, param2 ) {
   typeof functionName === 'function';
}
typeof functionName === 'function';
```

Pero también las podemos definir como expresiones, con la diferencia de que se oculta el nombre de la función con el scope exterior. El scope exterior se debe invocar utilizando una referencia a la función, que puede ser almacenada en una variable, devuelta o pasada como argumento:

```
var functionRef = function functionName( param, param2 ) {
   typeof functionName === 'function';
   // accessing the variable in the outer scope
   typeof functionRef === 'function';
}
typeof functionName === 'undefined';
typeof functionRef === 'function';
```

Como ya tenemos una manera de invocar una función sin necesidad de utilizar su nombre, podemos definirlo sin ella. Esto se conoce como ***funciones anónimas***:

```
var functionRef = function( param, param2 ) {
   // accessing the variable in the outer scope
   typeof functionRef === 'function';
}
typeof functionRef === 'function';
```

No hay necesidad de usar un nombre.