Funciones de JavaScript

Una función de JavaScript es un bloque de código diseñado para realizar una tarea en particular.

Se ejecuta una función JavaScript cuando "algo" lo invoca (lo llama).

Ejemplo

```
function myFunction(p1, p2) {
  return p1 * p2;              // The function returns the product of p1 and p2
}
```

Inténtalo tú mismo "

Sintaxis de la función JavaScript

Una función JavaScript se define con la palabra clave function , seguido de un nombre , seguido de paréntesis () .

Los nombres de las funciones pueden contener letras, dígitos, guiones bajos y signos de dólar (mismas reglas que variables).

Los paréntesis pueden incluir nombres de parámetros separados por comas: (parámetro1, parámetro2, ...)

El código a ejecutar, por la función, se coloca dentro de los paréntesis rizados: {}

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

Los parámetros de la función son los nombres que figuran en la definición de la función.

Los argumentos de función son los valores reales recibidos por la función cuando se invoca.

Dentro de la función, los argumentos (los parámetros) se comportan como variables locales.

Una función es muy similar a un procedimiento o una subrutina, en otros lenguajes de programación.

Invocación de funciones

El código dentro de la función se ejecutará cuando "algo" invoque (llame) la función:

- Cuando ocurre un evento (cuando un usuario hace clic en un botón)
- Cuando se invoca (se llama) desde el código JavaScript
- Automáticamente (auto invocada)

Retorno de función

Cuando JavaScript alcanza una declaración de retorno , la función dejará de ejecutarse.

Si la función se invocó desde una instrucción, JavaScript "regresará" para ejecutar el código después de la instrucción de invocación.

Las funciones a menudo calculan un valor de retorno . El valor devuelto se devuelve al "llamante":

Ejemplo

Calcule el producto de dos números y devuelva el resultado:

```
var   x = myFunction(4, 3);        // Function is called, return value will end
up in x

function myFunction(a, b) {
    return a * b;                // Function returns the product of a and b
}
```

El resultado en x será:

```
12
```

Inténtalo tú mismo "

El operador () invoca la función

Con el ejemplo siguiente, toCelsius se refiere al objeto de función y toCelsius () se refiere al resultado de la función.

El acceso a una función sin () devolverá la definición de la función en lugar del resultado de la función:

Ejemplo

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Inténtalo tú mismo "

Funciones utilizadas como valores variables

Las funciones se pueden usar de la misma manera que usa variables, en todos los tipos de fórmulas, asignaciones y cálculos.

Ejemplo

En lugar de usar una variable para almacenar el valor de retorno de una función:

```
var x = toCelsius(77);
var text = "The temperature is " + x + " Celsius";
```

Puede usar la función directamente, como un valor de variable:

```
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

Inténtalo tú mismo "

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 »

Funciones globales JavaScript

JavaScript cuenta con una serie de funciones predefinidas, es decir, funciones que ya están integradas en el lenguaje. Podemos utilizar estas funciones en cualquier momento sin tener que declararlas previamente y sin tener que conocer todas las instrucciones que realiza. Simplemente debemos conocer el nombre de la función y el resultado final que obtenemos al utilizarla.

Function	Description
deUcodeRI()	Decodes a URI
decodeURIComponent()	Decodes a URI component
encodeURI()	Encodes a URI
encodeURIComponent()	Encodes a URI component
escape()	Deprecated in version 1.5. Use encodeURI() or encodeURIComponent() instead
eval()	Evaluates a string and executes it as if it was script code
isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string
unescape()	Deprecated in version 1.5. Use decodeURI() or decodeURIComponent() instead

Funciones o Métodos?

Tiene sentido llamar a la lista anterior **funciones globales** en lugar de métodos globales porque las funciones se llaman globalmente y no cualquier objeto.

De todos modos, también puede llamar métodos a estas funciones, ya que son métodos del objeto global donde se ejecutan. En un navegador web, el objeto global es la ventana del navegador. Entonces isNaN() es en realidad un método window: window.isNaN().

JavaScript eval() Function

Example

Evaluate/Execute JavaScript code/expressions:

```
var   x = 10;
var y = 20;
var a = eval("x * y") + "<br>";
var b = eval("2 + 2") + "<br>";
var c = eval("x + 17") + "<br>";

var res = a + b + c;
```

The result of res will be:

```
200
4
27
```

Try it Yourself »

Definición y uso

La función eval () evalúa o ejecuta un argumento.

Si el argumento es una expresión, eval () evalúa la expresión. Si el argumento es una o más sentencias JavaScript, eval () ejecuta las sentencias.

JavaScript isFinite() Function

Example

Check whether a number is a finite, legal number:

```
var   a = isFinite(123) + "<br>";
var b = isFinite(-1.23) + "<br>";
var c = isFinite(5-2) + "<br>";
var d = isFinite(0) + "<br>";
var e = isFinite("123") + "<br>";
var f = isFinite("Hello") + "<br>";
var g = isFinite("2005/12/12");

var res = a + b + c + d + e + f + g;
```

The result of res will be:

```
true
true
true
true
true
false
false
```

Try it Yourself »

Definition and Usage

The isFinite() function determines whether a number is a finite, legal number.

This function returns false if the value is +infinity, -infinity, or NaN (Not-a-Number), otherwise it returns true.

JavaScript isNaN() Function

Example

Check whether a value is NaN:

```
isNaN(  123) //false
isNaN(-1.23) //false
isNaN(5-2) //false
isNaN(0) //false
isNaN('123') //false
isNaN('Hello') //true
isNaN('2005/12/12') //true
isNaN('') //false
isNaN(true) //false
isNaN(undefined) //true
isNaN('NaN') //true
isNaN(NaN) //true
isNaN(0 / 0) //true
```

Try it Yourself »

Definición y uso

La función isNaN () determina si un valor es un número ilegal (Not-a-Number).

Esta función devuelve true si el valor equivale a NaN. De lo contrario devuelve false.

Esta función es diferente del método numérico Number.isNaN () .

La función isNaN () global, convierte el valor probado en un número y lo prueba.

Number.isNaN () no convierte los valores en un número y no devuelve true para ningún valor que no sea del tipo Number.

JavaScript String() Function

Example

Convert different objects to strings:

```
var   x1 = Boolean(0);
var x2 = Boolean(1);
var x3 = new Date();
var x4 = "12345";
var x5 = 12345;

var res =
String(x1) + "<br>" +
String(x2) + "<br>" +
String(x3) + "<br>" +
String(x4) + "<br>" +
String(x5);
```

The result of res will be:

```
false
true
Tue Oct 10 2017 18:16:22 GMT+0200 (CEST)
12345
12345
```

Try it Yourself »

Definition and Usage

The String() function converts the value of an object to a string.

Note: The String() function returns the same value as toString() of the individual objects.

JavaScript Number() Function

Example

Convert different object values to their numbers:

```
var   x1 = true;
var x2 = false;
var x3 = new Date();
var x4 = "999";
var x5 = "999 888";

var n =
Number(x1) + "<br>" +
Number(x2) + "<br>" +
Number(x3) + "<br>" +
Number(x4) + "<br>" +
Number(x5);
```

The result of n will be:

```
1
0
1382704503079
999
NaN
```

Try it Yourself »

Definición y uso

La función Number () convierte el argumento del objeto en un número que representa el valor del objeto.

Si no se puede convertir el valor en un número legal, se devuelve NaN.

Nota: Si el parámetro es un objeto Date, la función Number () devuelve el número de milisegundos desde la medianoche del 1 de enero de 1970 UTC.

JavaScript parseInt() Function

Example

Analiza diferentes cadenas:

```
var   a = parseInt("10") + "<br>";
var b = parseInt("10.00") + "<br>";
var c = parseInt("10.33") + "<br>";
var d = parseInt("34 45 66") + "<br>";
var e = parseInt(" 60 ") + "<br>";
var f = parseInt("40 years") + "<br>";
var g = parseInt("He was 40") + "<br>";

var h = parseInt("10", 10)+ "<br>";
var i = parseInt("010")+ "<br>";
var j = parseInt("10", 8)+ "<br>";
var k = parseInt("0x10")+ "<br>";
var l = parseInt("10", 16)+ "<br>";

var n = a + b + c + d + e + f + g + "<br>" + h + i + j + k +l;
```

The result of n will be:

```
10
10
10
34
60
40
NaN

10
10
8
16
16
```

Try it Yourself »

Definición y uso

La función parseInt () analiza una cadena y devuelve un entero.

El parámetro radix se utiliza para especificar qué sistema numérico se utilizará, por ejemplo, una raíz de 16 (hexadecimal) indica que el número en la cadena debe ser analizado desde un número hexadecimal a un número decimal.

Si se omite el parámetro radix, JavaScript asume lo siguiente:

- Si la cadena comienza con "0x", la raíz es 16 (hexadecimal)
- Si la cadena comienza con "0", la raíz es 8 (octal). Esta característica está obsoleta
- Si la cadena comienza con cualquier otro valor, la raíz es 10 (decimal)

Nota: Sólo se devuelve el primer número de la cadena.

Nota: Se permiten espacios iniciales y finales.

Nota: Si el primer carácter no puede ser convertido en un número, parseInt () devuelve NaN.

Nota: Los navegadores más antiguos darán a parseInt ("010") como 8, ya que las versiones anteriores de ECMAScript (anteriores a ECMAScript 5, usan la raíz octal (8) como predeterminada cuando la cadena comienza con "0" .A partir de ECMAScript 5, la predeterminada es la raíz decimal (10).

JavaScript parseFloat() Function

Example

Parse different strings:

```
var   a = parseFloat("10") + "<br>";
var b = parseFloat("10.00") + "<br>";
var c = parseFloat("10.33") + "<br>";
var d = parseFloat("34 45 66") + "<br>";
var e = parseFloat(" 60 ") + "<br>";
var f = parseFloat("40 years") + "<br>";
var g = parseFloat("He was 40") + "<br>";

var n = a + b + c + d + e + f + g;
```

The result of n will be:

```
10
10
10.33
34
60
40
NaN
```

Try it Yourself »

Definición y uso

La función parseFloat() analiza una cadena y devuelve un número de punto flotante.

Esta función determina si el primer carácter de la cadena especificada es un número. Si lo es, analiza la cadena hasta que llega al final del número y devuelve el número como un número, no como una cadena.

Nota: Sólo se devuelve el primer número de la cadena.

Nota: Se permiten espacios iniciales y finales.

Nota: Si el primer carácter no se puede convertir en un número, parseFloat() devuelve NaN.

Definir funciones en JavaScript

Las funciones JavaScript se definen con la palabra clave function .

Puede utilizar una declaración de función o una expresión de función .

Declaraciones de funciones

Las funciones se declaran con la siguiente sintaxis:

```
function functionName(parameters) {
  code to be executed
}
```

Las funciones declaradas no se ejecutan inmediatamente. Son "guardados para uso posterior", y serán ejecutados más tarde, cuando se invocan (cuando son llamadas).

Ejemplo

```
function myFunction(a, b) {
    return a * b;
}
```

Inténtalo tú mismo "

Las comas se utilizan para separar las sentencias de JavaScript ejecutables.
Dado que una declaración de función no es una sentencia ejecutable, no es habitual terminarla con un punto y coma.

Expresiones de funciones

También se puede definir una función de JavaScript mediante una expresión .

Una expresión de función se puede almacenar en una variable:

Ejemplo

```
var   x = function (a, b) {return a * b};
```

Inténtalo tú mismo "

Después de que una expresión de función se haya almacenado en una variable, la variable se puede utilizar como una función:

Ejemplo

```
var   x = function (a, b) {return a * b};
var z = x(4, 3);
```

Inténtalo tú mismo "

La función anterior es en realidad una función anónima (una función sin un nombre).

Las funciones almacenadas en variables no necesitan nombres de función. Siempre se invoca (llamado) utilizando el nombre de la variable.

La función anterior termina con un punto y coma porque es una parte de una sentencia ejecutable.

Función hoisting

Hoisting es el comportamiento predeterminado de JavaScript de las declaraciones moviéndolas a la parte superior del ámbito actual.

El hoisting se aplica a declaraciones de variables y a declaraciones de funciones.

Debido a esto, las funciones de JavaScript se pueden llamar antes de que se declaren:

```
myFunction(5);
```

```
function myFunction(y) {
    return y * y;
}
```

Las funciones definidas mediante una expresión no se “adelantan”.

Las funciones son objetos

El operador typeof en JavaScript devuelve "function" para funciones.

Sin embargo, las funciones JavaScript se pueden describir mejor como objetos.

Las funciones JavaScript tienen propiedades y métodos .

La propiedad arguments.length devuelve el número de argumentos recibidos cuando se invocó la función:

Ejemplo

```
function myFunction(a, b) {
    return arguments.length;
}
```

Inténtalo tú mismo "

El método toString () devuelve la función como una cadena:

Ejemplo

```
function myFunction(a, b) {
    return a * b;
}

var txt = myFunction.toString();
```

Inténtalo tú mismo "

Una función definida como la propiedad de un objeto, se llama un método al objeto.
Una función diseñada para crear nuevos objetos, se denomina constructor de objetos.

Parámetros de funciones JavaScript

Una función JavaScript no realiza ninguna comprobación de los valores de los parámetros (argumentos).

Parámetros de Función y Argumentos

```
functionName (parameter1, parameter2, parameter3) {
    code to be executed
}
```

Los parámetros de función son los nombres enumerados en la definición de la función.

Los argumentos de función son los valores reales pasados a (y recibidos por) la función.

Reglas de parámetros

Las definiciones de funciones JavaScript no especifican tipos de datos para los parámetros.

Las funciones JavaScript no realizan la comprobación de tipo en los argumentos pasados.

Las funciones JavaScript no comprueban el número de argumentos recibidos.

Parámetros predeterminados

Si se llama a una función con argumentos perdidos (menos que los declarados), los valores que faltan se establecen en: undefined

A veces esto es aceptable, pero a veces es mejor asignar un valor predeterminado al parámetro:

Ejemplo

```
function myFunction(x, y) {
    if (y === undefined) {
        y = 0;
    }
}
```

Inténtalo tú mismo "

Si se llama a una función con demasiados argumentos (más que declarados), estos argumentos pueden ser alcanzados utilizando el objeto arguments .

El objeto arguments

Las funciones JavaScript tienen un objeto incorporado denominado objeto argumentos.

El objeto argumento contiene una matriz de los argumentos utilizados cuando se invocó la función (invocada).

De esta manera usted puede simplemente usar una función para encontrar (por ejemplo) el valor más alto en una lista de números:

Ejemplo

```
x = findMax(  1, 123, 500, 115, 44, 88);

function findMax() {
    var i;
    var max = -Infinity;
    for (i = 0; i < arguments.length; i++) {
        if (arguments[i] > max) {
            max = arguments[i];
        }
    }
    return max;
}
```

Inténtalo tú mismo "

O cree una función para sumar todos los valores de entrada:

Ejemplo

```
x = sumAll(  1, 123, 500, 115, 44, 88);

function sumAll() {
    var i, sum = 0;
    for (i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    return sum;
}
```

Inténtalo tú mismo "

Los argumentos se pasan por valor

Los parámetros, en una llamada de función, son los argumentos de la función.

Los argumentos de JavaScript se pasan por valor : la función sólo conoce los valores, no las ubicaciones del argumento.

Si una función cambia el valor de un argumento, no cambia el valor original del parámetro.

Los cambios en los argumentos no son visibles (reflejados) fuera de la función.

Los objetos se pasan por referencia

En JavaScript, las referencias de objeto son valores.

Debido a esto, los objetos se comportarán como si fueran pasados por referencia:

Si una función cambia una propiedad de objeto, cambia el valor original.

Los cambios en las propiedades del objeto son visibles (reflejados) fuera de la función.

JavaScript Function Invocation

El código dentro de una función JavaScript se ejecutará cuando "algo" lo invoque.

Invocación de una función JavaScript

El código dentro de una función no se ejecuta cuando se define la función .

El código dentro de una función se ejecuta cuando se invoca la función .

Es común utilizar el término " llamar a una función " en lugar de " invocar una función ".

También es común decir "llamar a una función", "iniciar una función" o "ejecutar una función".

En este tutorial, usaremos invocar , porque una función JavaScript puede ser invocada sin ser llamada.

Invocación de una función como función

Ejemplo

```
function myFunction(a, b) {
    return a * b;
}
myFunction(10, 2);          // Will return 20
```

Inténtalo tú mismo "

La función anterior no pertenece a ningún objeto. Pero en JavaScript siempre hay un objeto global predeterminado.

En HTML el objeto global predeterminado es la página HTML en sí, por lo que la función anterior "pertenece" a la página HTML.

En un navegador, el objeto de página es la ventana del navegador. La función anterior se convierte automáticamente en una función de ventana.

myFunction () y window.myFunction () es la misma función:

Ejemplo

```
function   myFunction(a, b) {
    return a * b;
}
window.myFunction(10, 2);    // Will also return 20
```

Inténtalo tú mismo "

Esta es una forma común de invocar una función JavaScript, pero no una práctica muy buena.
Las variables, métodos o funciones globales pueden crear fácilmente conflictos de nombres y errores en el objeto global.

La palabra clave this

En JavaScript, lo que se llama this, es el objeto que "posee" el código actual.

El valor de this, cuando se utiliza en una función, es el objeto que "posee" la función.

Tenga en cuenta que this no es una variable. Es una palabra clave. No puede cambiar el valor de this .

El objeto global

Cuando se llama a una función sin un objeto propietario, el valor de this se convierte en el objeto global.

En un navegador web, el objeto global es la ventana del navegador.

Este ejemplo devuelve el objeto window como el valor de this :

Ejemplo

```
function myFunction() {
    return this;
}
myFunction();               // Will return the window object
```

Inténtalo tú mismo "

Invocar una función como una función global, hace que el valor de **this** sea el objeto global.