**Definiendo funciones**

La definición de una función consiste en la palabra clave '**function**' seguida de:

* El nombre de la función.
* Una lista de argumentos para la función, encerrados entre paréntesis y separados por comas.
* Las sentencias de JavaScript que definen la función encerradas entre llaves {}.

Estas sentencias incluidas en la función pueden incluir llamadas a otras funciones que estén definidas en la aplicación.

Por ejemplo, el siguiente código define una sencilla función llamada 'cuadrado'.

function cuadrado(número) {

return número\*número

}

La función cuadrado recibe un argumento, llamado 'número'. La función consiste en una sentencia que retorna el argumento de la función multiplicado por sí mismo. La declaración '**return**' especifica el valor devuelto por la función.

return número\*número

Todos **los parámetros** son pasados a las funciones como **valores**;

* El valor es pasado a la función, pero si la función cambia el valor del parámetro, este cambio no es reflejado globalmente o en la función llamada.
* Sin embargo, **si pasa un objeto** como parámetro a una función y la función cambia las propiedades del objeto, **este cambio es visible fuera de la función** como se muestra en el siguiente ejemplo:

function miFuncion(elObjecto) {

elObjecto.fabricante="Toyota"

}

micarro = {fabricante:"Honda", modelo:"Accord", año:1998};

x=micarro.fabricante; // devuelve Honda

miFuncion(micarro); // pasa el objeto 'micarro' a la función

y=micarro.fabricante; // devuelve 'Toyota' (la propiedad es cambiada por la función)

Una función puede ser definida basada en una condición. Por ejemplo, dada la siguiente definición de función:

if (número == 0)

{

function miFuncion(elObjecto) {

elObjecto.fabricante="Toyota"

}

}

La función 'miFuncion' será definida únicamente si la variable 'número' es igual a 0. Si 'número' no es igual a 0, la función no será definida y cualquier intento de ejecutarla fracasará.

**Una función puede también ser definida dentro de una expresión**, esta es llamada una expresión funcional, **típicamente tal función** es **anónima**; no tiene que tener un nombre, por ejemplo, la función 'cuadrado' podría haber sido definida como:

const cuadrado = function(número) {return número \* número};

Esto es recomendable cuando se pasa una función **como** **argumento** a otra función.

El siguiente ejemplo muestra la función 'mapa' siendo definida y después llamada, y con una **función** anónima como su primer parámetro.

function mapa(función,arreglo\_valores) {

var resultado=new Array;

for (var i = 0; i != arreglo\_valores.length; i++) //cuantos elementos

//pasan - 1?

resultado[i] = función(arreglo[i]); //asigne valores del arreglo como

//parámetros a la función

return resultado; //retorna arreglo\_resultado al invocar a la función por

//cada elemento del arreglo

}

La llamada:

mapa(function(x) {return x \* x \* x}, [0, 1, 2, 5, 10]);

Función anónima

Retornará

[0, 1, 8, 125, 1000]

### Llamando a las funciones

**Definir una función no la ejecuta**, definir una función simplemente la nombra y especifica qué debe hacer cuando dicha función es invocada. Una invocación a la función lo que hace es realizar las acciones especificadas con los parámetros indicados.

Por ejemplo, si ya se ha definido la función 'cuadrado', la podría invocar como sigue:

cuadrado(5)

Esta declaración invoca a la función 'cuadrado' con el argumento '5'. La función ejecutaría sus declaraciones y devolvería el valor '25'

Los argumentos de una función no están limitados a cadenas o números, puede también pasarse objetos enteros a ésta. La función 'mostrar\_propiedades' (definida en <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Trabajando_con_objectos>) es un buen ejemplo de una función que recibe un objeto como argumento.

function mostrarPropiedades(objeto, nombreObjeto) {

var resultado = "";

for (var i in objeto) {

if (objeto.hasOwnProperty(i)) {

resultado += nombreObjeto + "." + i + " = " + objeto[i] + "\n";

}

}

return resultado;

}

Una función puede incluso ser recursiva, esto quiere decir que se puede llamar a sí misma. Como ejemplo aquí se tiene una función que calcula factoriales llamándose a sí misma:

function factorial(numero) {

if ((numero == 0) || (numero == 1))

return 1;

else {

var resultado = (numero \* factorial(numero-1));

return resultado;

}

}

Entonces se podrían calcular los factoriales del 1 al 5 como se muestra aquí abajo:

a=factorial(1); // devolvera 1

b=factorial(2); // devolvera 2

c=factorial(3); // devolvera 6

d=factorial(4); // devolvera 24

e=factorial(5); // devolvera 120

Funciones predefinidas

JavaScript tiene varias funciones predefinidas de nivel superior:

* [Función eval](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Gu%c3%ada_JavaScript_1.5/Funciones_predefinidas/Funci%c3%b3n_eval)
* [Función isFinite](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Gu%c3%ada_JavaScript_1.5/Funciones_predefinidas/Funci%c3%b3n_isFinite)
* [Función isNaN](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Gu%c3%ada_JavaScript_1.5/Funciones_predefinidas/Funci%c3%b3n_isNaN)
* [Funciones parseInt y parseFloat](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Gu%c3%ada_JavaScript_1.5/Funciones_predefinidas/Funciones_parseInt_y_parseFloat)
* [Funciones Number y String](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Gu%c3%ada_JavaScript_1.5/Funciones_predefinidas/Funciones_Number_y_String)
* [Funciones escape y unescape](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Gu%c3%ada_JavaScript_1.5/Funciones_predefinidas/Funciones_escape_y_unescape)

### Función eval

La función **eval** evalúa una cadena de código JavaScript sin referencia a ningún objeto en particular. La sintaxis de eval es como sigue:

eval(expresión)

Donde expresión es la cadena a ser evaluada.

Si la cadena representa una expresión, eval evaluará la expresión. Si el argumento representa una o más declaraciones JavaScript, eval realizará las declaraciones. No invoque a eval para evaluar expresiones aritméticas, JavaScript evalúa expresiones aritméticas de forma automática

### Función isFinite

La función [isFinite](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/Funciones_predefinidas/es/Referencia_de_JavaScript_1.5/Funciones_globales/isFinite) **evalúa un argumento para determinar si éste es un número finito**. La sintaxis de isFinite es como sigue:

isFinite(número)

Donde número sería el número a evaluar.

Si el argumento es NaN, infinito positivo o infinito negativo, este método devolverá el valor false, en el resto de los casos devolverá true.

El siguiente código comprueba EntradaCliente para determinar si este es un número finito.

if(isFinite(EntradaCliente))

{

/\* realizar las declaraciones que sean \*/

}

### Función isNaN

La función [isNaN](https://developer.mozilla.org/es/Referencia_de_JavaScript_1.5/Funciones_globales/isNaN" \o "es/Referencia_de_JavaScript_1.5/Funciones_globales/isNaN) evalúa un argumento para determinar si éste no es un número (isNaN [not a number]). La sintaxis para isNaN es:

isNaN(Valor\_a\_probar)

Donde Valor\_a\_probar es el valor que desea evaluar.

Las funciones parseFloat y parseInt retornan '**NaN**' cuando evalúan un valor que no es un número.

El siguiente código evalúa ValorFlotante para determinar si es un número y entonces llamar a una función en consecuencia del resultado.

ValorFlotante=parseFloat(convertiraFlotante)

if (isNaN(ValorFlotante)) {

notFloat()

} else {

isFloat()

}

### Funciones parseInt and parseFloat

Las dos funciones "convertidoras", [parseInt](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Referencia_de_JavaScript_1.5/Funciones_globales/parseInt) y [parseFloat](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Referencia_de_JavaScript_1.5/Funciones_globales/parseFloat), retornan un valor numérico cuando se pasa una cadena como un argumento.

La sintaxis de **parseFloat** es:

parseFloat(cadena)

donde parseFloat convierte su argumento, la cadena cadena, e intenta retornar un número de punto flotante. Si encuentra un caracter diferente a un signo (+ o -), un número (0-9), un punto decimal, o un exponente, entonces retorna el valor hasta antes del punto e ignora este caracter y todos los caracteres sucesivos. Si el primer carácter no puede ser convertido a un número, retorna "NaN" (no es número).

La sintaxis de **parseInt** es

parseInt(cadena [, raíz])

parseInt convierte su primer argumento, la cadena cadena, e intenta retornar un entero en una base de raíz especificada, indicada por el segundo parámetro opcional, radix. Por ejemplo, una raíz de 10 indica convertir a número decimal, ocho a octal, dieciséis a hexadecimal y así sucesivamente. Para raíces mayores a diez, las letras del alfabeto indican números mayores a nueve. Por ejemplo, para números hexadecimales (base 16), son utilizadas desde la A hasta la F.

Si parseInt encuentra un carácter que no es un número en la raíz especificada, lo ignora y todos los caracteres sucesivos y retorna el valor entero convertido hasta el punto. Si el primer caracter no puede ser convertido a un número en una raíz especificada, retorna "NaN." La función parseInt trunca la cadena a valores enteros.

### Funciones Number y String

Las funciones Number y String te permiten convertir un objeto a un número o cadena. La sintaxis de estas funciones es:

Number(objRef)

String(objRef)

donde objRef es un objeto de referencia.

El siguiente ejemplo convierte el objeto [Date](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Gu%c3%ada_JavaScript_1.5/Objetos_base_predefinidos/Objeto_Date) a una cadena.

D = new Date (430054663215)

// Retorna lo siguiente

// "Thu Aug 18 04:37:43 GMT-0700 (Pacific Daylight Time) 1983"

x = String(D)

### Función encodeURI

Codifica un Identificador de Recurso Uniforme (Uniform Resource Identifier, URI) reemplazando cada instancia de ciertos caracteres por uno, dos o tres secuencias de escape representando el carácter codificado en UTF-8.

Sintaxis: encodeURI(URI)

### Parámetros

**URI**

Un Identificador de Recurso Uniforme codificado.

Descripción:

Se asume que el URI es un URI completo, a fin de no codificar caracteres reservados que tienen un significado especial en la URI. encodeURI reemplaza todos los caracteres, excepto los siguientes, con las secuencias de escape UTF-8:

|  |  |
| --- | --- |
| Tipo | Incluye |
| Caracteres reservados | ; , / ? : @ & = + $ |
| Caracteres No escape | Alfabéticos, dígitos decimales, - \_ . ! ~ \* ' ( ) |
| Almohadilla | # |

Tenga en cuenta que encodeURI por sí sola no puede formar solicitudes correctas HTTP, GET y POST, como por XMLHTTPRequests, porque "&", "+", y "=" no están codificados, los cuales son tratados como caracteres especiales en las solicitudes GET y POST. [encodeURIComponent](https://developer.mozilla.org/es/Core_JavaScript_1.5_Reference/Global_Functions/encodeURIComponent), sin embargo, codifica estos caracteres. Lo más probable es que estos comportamientos no sean compatibles en los navegadores Web.

**Function decodeURI**

Decodifica un Identificador de Recurso Uniforme (Uniform Resource Identifier, URI) previamente creado por [encodeURI](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/encodeURI" \o "Codifica un Identificador de Recurso Uniforme (Uniform Resource Identifier, URI) reemplazando cada instancia de ciertos carácteres por uno, dos o tres secuencias de escape representando el carácter codificado en UTF-8.) o por una rutina similar.

**Sintaxis:** decodeURI(encodedURI)

### Parámetros

**encodedURI**

Un completo, Identificador de Recurso Uniforme.

**Descripción:**

Sustituye a cada secuencia de escape codificado en URI con el carácter que representa.

No decodifica secuencias de escape que no se puedan haber introducido por [encodeURI](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/encodeURI" \o "Codifica un Identificador de Recurso Uniforme (Uniform Resource Identifier, URI) reemplazando cada instancia de ciertos carácteres por uno, dos o tres secuencias de escape representando el carácter codificado en UTF-8.).

**Function** encodeURIComponent()

The **encodeURIComponent()** method encodes a Uniform Resource Identifier (URI) component by replacing each instance of certain characters by one, two, three, or four escape sequences representing the UTF-8 encoding of the character (will only be four escape sequences for characters composed of two "surrogate" characters).

**Syntax**:

encodeURIComponent(str);

### Parameters

**str**

String. A component of a URI.

**Description:**

encodeURIComponent escapes all characters except the following: alphabetic, decimal digits, - \_ . ! ~ \* ' ( )

Note that an [URIError](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/URIError" \o "The documentation about this has not yet been written; please consider contributing!) will be thrown if one attempts to encode a surrogate which is not part of a high-low pair, e.g.,

// high-low pair ok

alert(encodeURIComponent('\uD800\uDFFF'));

// lone high surrogate throws "URIError: malformed URI sequence"

alert(encodeURIComponent('\uD800'));

// lone low surrogate throws "URIError: malformed URI sequence"

alert(encodeURIComponent('\uDFFF'));

To avoid unexpected requests to the server, you should call encodeURIComponent on any user-entered parameters that will be passed as part of a URI. For example, a user could type "Thyme &time=again" for a variable comment. Not using encodeURIComponent on this variable will give comment=Thyme%20&time=again. Note that the ampersand and the equal sign mark a new key and value pair. So instead of having a POST comment key equal to "Thyme &time=again", you have two POST keys, one equal to "Thyme " and another (time) equal to again.

For [application/x-www-form-urlencoded](http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#application/x-www-form-urlencoded-encoding-algorithm) (POST), spaces are to be replaced by '+', so one may wish to follow a encodeURIComponent replacement with an additional replacement of "%20" with "+".

To be more stringent in adhering to [RFC 3986](http://tools.ietf.org/html/rfc3986) (which reserves !, ', (, ), and \*), even though these characters have no formalized URI delimiting uses, the following can be safely used:

function fixedEncodeURIComponent (str) {

return encodeURIComponent(str).replace(/[!'()]/g, escape).replace(/\\*/g, "%2A");

}

## **Function decodeURIComponent**

## Sumario

Decodifica un Identificador de Recurso Uniforme (Uniform Resource Identifier, URI) previamente creado por [encodeURIComponent](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/encodeURIComponent) o por una rutina similar.

## Sintaxis:

## decodeURIComponent(encodedURI)

## Parámetros:

## encodedURI

## Un Identificador de Recurso Uniforme codificado.

## Descripción

Sustituye a cada secuencia de escape codificado en URI con el carácter que representa.

### Usando el objeto arguments

Una cosa llamativa para los programadores que se inician en JavaScript, es que los parámetros pasados a una función, no tienen por qué definirse a priori, en número. Es decir, se puede definir una función, con un parámetro y al usar la función pasarle varios. Todos estos parámetros son recibidos como se indica a continuación.

Los argumentos de una función son guardados en un objeto como si fuera un Arreglo.

Dentro de una función puede **localizar los argumentos pasados** de la siguiente forma:

arguments[i]

Donde i sería el número de orden del argumento, empezando desde el cero (0). Así que el primer argumento pasado a la función sería arguments[0]. El número total de argumentos sería indicado por

arguments.length

Utilizando el objeto arguments, se puede invocar a una función con más argumentos de los que ésta esté formalmente declarada. Esto es a menudo muy útil si no se sabe con antelación cuantos argumentos serán pasados a la función. Puede utilizar arguments.length para determinar el número de argumentos realmente pasados a la función y entonces tratar cada argumento usando el objeto arguments

Por ejemplo, considerando una función que une varias cadenas, el único argumento formal para la función es una cadena que especifica los caracteres que separan los elementos a unir. La función es definida así:

function miConcatenacion(separador) {

var resultado = ""; // inicia la lista

// Interactúa a través de los argumentos

for (var i = 1; i < arguments.length; i++) //a partir del argumento 1 en

//adelante, argumento 0 es separador

{

resultado += arguments[i] + separador;

}

return resultado;

}

Puede pasar cualquier número de argumentos a esta función y ésta concatenará cada argumento a la lista de cadenas.

// devuleve "rojo, naranja, azul,"

miConcatenacion(", ", "rojo", "naranja", "azul"); //separador y valores

// devolverá "elefante; jirafa; león; chita; "

miConcatenacion("; ", "elefante", "jirafa", "león", "chita");

// devolverá "sgae. chorizos. ladrones. rufianes. macacos. "

miConcatenacion(". ", "sgae", "chorizos", "ladrones", "rufianes", "macacos");

(Si desea consultar [*Objeto Function*](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5/es/Referencia_de_JavaScript_1.5/Objetos_globales/Function)en la referencia del núcleo de JavaScript para obtener más información.)

### JavaScript 1.3 y versiones anteriores

El objeto arguments es una propiedad del objeto Function y puede ser precedido del nombre de la función.

ejemplo: Nombrefunción.arguments[i]