**Resumen JavaScript**

**Kenia Montano Granados**

Contenido

[¿Qué es JavaScript? 3](#_Toc410987845)

[JavaScript y Java 3](#_Toc410987846)

[Palabras Reservadas en JavaScript 3](#_Toc410987847)

[Futuras Palabras Reservadas en JavaScript. 5](#_Toc410987848)

[toLowerCase() 5](#_Toc410987849)

[toUpperCase() 5](#_Toc410987850)

[alert() 6](#_Toc410987851)

[prompt() 7](#_Toc410987852)

[eval() 8](#_Toc410987853)

[confirm() 8](#_Toc410987854)

[open() 9](#_Toc410987855)

[close() 9](#_Toc410987856)

[Programación Orientada a Objeto 9](#_Toc410987857)

[Conceptos 9](#_Toc410987858)

[¿Se Puede Guardar una función en una variable? 10](#_Toc410987859)

[Diferencia entre $(document).ready() y window. onload 11](#_Toc410987860)

[Onunload 11](#_Toc410987861)

[<noscript> 11](#_Toc410987862)

[Typeof 12](#_Toc410987863)

[Null 12](#_Toc410987864)

[Undefined 12](#_Toc410987865)

[Diferencia entre == y === 13](#_Toc410987866)

[JavaScript The Good Parts 14](#_Toc410987867)

[Capitulo 1-Good Parts 14](#_Toc410987868)

[Capítulo 2-Grammar 14](#_Toc410987869)

[Capítulo 3-Objects 15](#_Toc410987870)

[Apéndice A-Awful Parts 16](#_Toc410987871)

[Date() 18](#_Toc410987872)

[Propiedades Date 18](#_Toc410987873)

[JavaScript Tipo de Conversion 19](#_Toc410987874)

[La Propiedad Constructor 20](#_Toc410987875)

[JavaScript Izar 20](#_Toc410987876)

[El Use Strict 21](#_Toc410987877)

[Errores Comunes en JavaScript 21](#_Toc410987878)

[parseInt: 24](#_Toc410987879)

[parseFloat 25](#_Toc410987880)

[indexOf 26](#_Toc410987881)

# ¿Qué es JavaScript?

Es un lenguaje de programación orientada a objetos, basada en prototipos que permite mejoras en el interfaz del usuario y páginas web dinámicas.

Una página dinámica se refiere al movimiento de textos, imágenes, animaciones o acciones que se realicen al presionar un botón o al hacer clic en la página.

Es un lenguaje de programación que permite a los desarrolladores crear acciones en su página web.

Los programas escritos por JavaScript se pueden probar desde el navegador sin necesidad de procesos intermedios.

# JavaScript y Java

JavaScript y Java son similares en algunos puntos. JavaScript tiene sintaxis, convenciones de nombres y controles básicos de flujo parecidos a los de Java, por eso se le cambio el nombre de Livescript a JavaScript. JavaScript soporta un sistema en tiempo de ejecución basado en un pequeño número de tipos de datos que representan valores numéricos, booleanos y cadenas. Tiene un modelo de objetos basado en prototipos este provee herencia dinámica.

# Palabras Reservadas en JavaScript

Las palabras reservadas son aquellas que no se pueden utilizar como identificadores ya que tienen un lenguaje específico utilizado para crear una función las cuales son:

break: permite terminar de forma abrupta un bucle.

case: pertenece al switch y este hace referencia a cada opción(caso) que se debe evaluar hasta que se cumpla alguna opción(caso) y se realizaría la función correspondiente.

catch: permite manejar el error.

continue: permite saltarse algunas repeticiones del bucle.

default: permiten parámetros formales para inicializar con valores por defecto si no hay valor o indefinido se pasa

delete:  retira una propiedad de un objeto.

do: es una variante del bucle while. Este bucle se ejecutará el bloque de código una vez, antes de comprobar si la condición es verdadera, entonces se repetirá el bucle mientras la condición es verdadera.

else: para especificar un bloque de código que se ejecutará, si la misma condición es falsa.

finally: permite ejecutar código, después de tratar de atrapar, sin importar el resultado.

for: crea un bucle que consiste en tres expresiones opcionales.

function: cada función en JavaScript es actualmente un objeto Function.

if: sirve para especificar un bloque de código que se ejecutará, si la condición especificada sea verdadera.

in: retorna true si la propiedad especificada está en el objeto especificado.

instanceof: devuelve verdadero si el objeto especificado es del tipo especificado.

new: crea una instancia de un tipo de objeto a partir de una función constructora nativa ó definida por el usuario.

return: especifica el valor devuelto por una función.

switch: sirve para especificar muchos bloques alternativos de código para ser ejecutado.

this: el valor de this está determinado por cómo se llama a la función. No puede ser establecida por una asignación en tiempo de ejecución, y esto puede ser diferente cada vez que la función es llamada.

try: permite probar un bloque de código para los errores.

throw: permite crear errores personalizados.

typeof: devuelve una cadena que indica el tipo del operando sin evaluarlo.

var: declaración de una variable, opcionalmente inicializada a un valor.

void:  especifica una expresión que se evalúa sin devolver un valor.

while: crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera. Dicha condición es evaluada antes de ejecutar la sentencia.

with: se extiende la cadena de ámbito de un comunicado.

# Futuras Palabras Reservadas en JavaScript.

Estas palabras no se deben utilizar como identificadores, aunque no posean ahora una función ya que en el futuro se utilizaran. Estas palabras son:

* class
* enum
* export
* extends
* import
* super

# toLowerCase()

Pasa una palabra en mayúscula a minúscula.

**Ejemplo:**

var str = "Hello World!";  
var res = str.toLowerCase();

El resultado de res será: hello world!

**Explicación:**

Lo que hace es que crea una variable (str) con alguna palabra escrita en mayúscula y en otra variable (res) se utiliza la variable de la palabra (str) y el toLowerCase() y esto hace que la palabra almacenada en str cambie a minúscula en la variable res.

# toUpperCase()

Hace lo contrario a toLowerCase(), es decir, pasa de minúscula a mayúscula.

**Ejemplo:**

var str = "Hello World!";  
var res = str.toUpperCase();

El resultado de res será: HELLO WORLD!

**Explicación:**

En este ejemplo pasa exactamente lo mismo al ejemplo anterior con la variante que la palabra va a pasar a mayúscula y no minúscula.

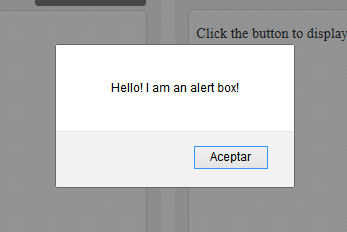
# alert()

Esta muestra un cuadro de alerta con el mensaje que este dentro de los paréntesis. Los mensajes pueden ser un string o una variable que contenga información.

**Ejemplo 1:**

alert("Hello! I am an alert box!!");

El resultado de este alert sera:



**Explicación:**

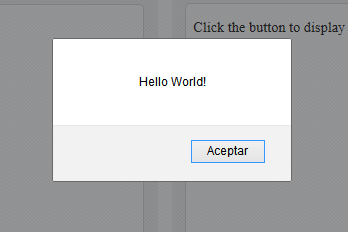
Es muy sencillo solamente aparece el texto que esta dentro del paréntesis.

**Ejemplo 2:**

var str = "Hello World!";

alert(str);

El resultado de este alert será:



**Explicación:**

Es lo mismo al ejemplo anterior solamente que utilizamos una variable para almacenar el mensaje.

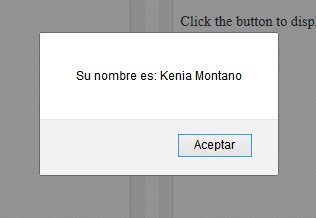
También podemos mezclar ambos ejemplos.

**Ejemplo 3:**

var str = "Kenia Montano";

alert(“Su nombre es: ”+ str);

El resultado sera:



**Explicación:**

Lo que hacemos es concatenar el texto del paréntesis con una variable (str) ya definida con un “+”.

# prompt()

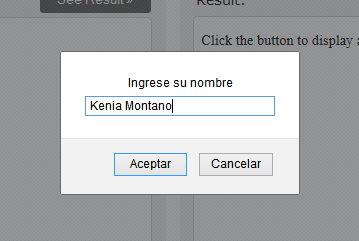
Este muestra un mensaje pidiendo el usuario que ingrese un dato (nombre, edad, nacionalidad, color, etc.) al ingresar lo que se pide, saldrá un mensaje, este mensaje dependerá de la información suministrada.

Este se puede almacenar en una variable para luego se utilice la variable en el mensaje, esta variable tendrá el dato introducido por el usuario.

**Ejemplo:**

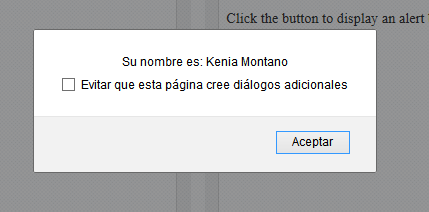
var str = prompt(“Ingrese su nombre”);

El resultado del prompt será:



alert(“Su nombre es: “+str);

El resultado del alert será:

**Explicación:**

El prompt() solicita el nombre y lo almacena en la variable luego por medio de un alert() se imprime el mensaje con el texto del alert mas el dato almacenado en la variable que fue ingresado por el usuario.

# eval()

Evalúa o ejecuta la expresión que se encuentre dentro del paréntesis.

**Ejemplo:**

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;

El resultado será:

a=200  
b=4  
c=27

res=231

**Explicación:**

Se crean primero dos variables (x,y) donde se les da un valor (10,20) y estas mismas variables se utilizan para realizar una multiplicación en la variable a (10 \* 20=200), en la variable b solamente da los números y la operación se realiza con eval() y en variable c (10 + 17=27) todas estas operaciones la realiza la función eval(). En res solamente se suma a + b +c (200+4+27=231).

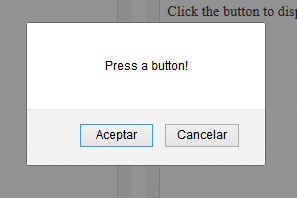
# confirm()

Este muestra un mensaje de confirmación con dos botone ok y cancelar, es utilizado para que el usuario verifique o acepte alguna función.

**Ejemplo:**

confirm("Press a button!");

El resultado de confirm será:



**Explicacion:**

El confirm solamente tira este mensaje con ok y con un cancel para que el usuario escoja.

# open()

Abre una nueva ventana en el navegador para que el usuario realice alguna función opcionalmente se le puede dar una url para que esta se dirija al página.

# close()

Hace lo contrario a open, es decir, cierra la ventana actual.

# Programación Orientada a Objeto

Un objeto posee estado, comportamiento e identidad.

**Estado:** Datos e información del objeto.

**Comportamiento**: Es la respuesta a los métodos o mensajes que se le den, es decir, lo que el objeto puede realizar.

**Identidad**: Es la propiedad que diferencia al objeto de los demás.

## Conceptos

**Clase:** Define las propiedades y comportamiento de un objeto.

**Herencia:** Cuando un objeto hereda atributos y operaciones de otro.

**Objeto:** Este puede poseer datos y métodos que reaccionen a eventos. Pueden asociarse a objetos reales o objetos internos del sistema.

**Método:** Lo que el objeto puede hacer.

**Evento:** La reacción que desencadena un objeto.

**Atributos:** Características.

**Mensaje:** Es el medio en que se comunican los objetos donde se ordena alguna ejecución.

**Propiedad o Atributo:** Contiene los datos asociados a un objeto.

**Componentes de un objeto:** Atributos, identidad, relaciones y métodos.

**Identificación de un objeto:** Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

**Constructor:** El constructor es llamado en el momento de la creación de la instancia (el momento en que se crea la instancia del objeto). El constructor es un método de la clase. El constructor se usa para establecer las propiedades del objeto o para llamar a los métodos para preparar el objeto para su uso.

# ¿Se Puede Guardar una función en una variable?

Si se puede guardar una función en una variable para luego utilizar el valor retornado por la función, si no se guarda en una variable el valor retornado de la función se pierde, es decir, no es utilizado.

**Ejemplo:**

function calculaPrecioTotal(precio, porcentajeImpuestos) {

var gastosEnvio = 10;

var precioConImpuestos = (1 + porcentajeImpuestos/100) \* precio;

var precioTotal = precioConImpuestos + gastosEnvio;

return precioTotal;

}

var precioTotal = calculaPrecioTotal(23.34, 16);

**Explicación:**

Lo primero que hace es crear una función que introduce por parámetros precio, porcentajeImpuestos y dentro de la función realiza las operaciones al final retorna (return) el valor que se encuentra en la variable precioTotal y ese va a ser el resultado de la función, fuera de la función se crea otra variable precioTotal donde se almacena la función que le da los valores a los parámetros calculaPrecioTotal(23.34, 16). Despues de esto podríamos utilizar sin ningún problema la variable precioTotal ya que esta contiene el resultado de la función.

# Diferencia entre $(document).ready() y window. onload

**$(document).ready:**

El .ready() cuando DOM(Document Obect Model) ha cargado.

Como este evento se da cuando se carga todo el documento es un buen momento para que se carguen los eventos y funciones de jQuery.

**window. onload:**

Este ejecuta un script cuando la página web carga completamente el contenido incluyendo imágenes, archivos de comandos, archivos CSS, etc.

El onload también se utiliza para comprobar el tipo y versión de navegador del visitante para mostrar la información correcta y se puede utilizar para hacer frente a los cookies.

# Onunload

Este ocurre cuando el usuario se desplaza a otra página por ejemplo al hacer click a un enlace, envía un formulario, cierra la ventana del navegador.

Sintaxis en HTML:

<*element* onunload="*myScript*">

Sintaxis en JavaScript:

object.onunload=function(){*myScript*};

# <noscript>

Define un contenido alternative para los usuarios tienen problemas con los comandos del navegador o que el navegador no admita script.

Es un elemento que se puede usar en el <head> y <body>.

Este se activara siempre que el navegador no admita scipt o que se desactiven.

# Typeof

Da el tipo de variable, es decir, según su valor da si es de string (texto), number (numero), boolean(true o false) , function(función) u object.

El typeof solo retornara el tipo de variable que corresponda.

**Ejemplo:**

typeof "John"                // Returns string   
typeof 3.14                  // Returns number  
typeof false                 // Returns boolean  
typeof [1,2,3,4]             // Returns object  
typeof {name:'John', age:34} // Returns object

typeof function(){}  // Returns function

**Explicación:**

Es muy sencillo a cada dato se le antepone un typeof y en la parte de comentario (//) solamente dice que clase de variable es.

# Null

Null en JavaScript es nada. Es algo que no existe pero aun asi corresponde en JavaScript a un objeto.

**Ejemplo:**

var person = null;    // Value is null, but type is still an object

**Explicación:**

Como pueden ver el valor es nulo pero el tipo es un object.

# Undefined

Es cuando una variable no posee valor. Y el typeof es undefined.

**Ejemplo:**

var person;              // Value is undefined, type is undefined

# Diferencia entre == y ===

**==**

Cuando se igualan dos datos y estos no poseen el mismo tipo pero si el mismo dato.

**Ejemplo:**

var num1 = “25”;

var num2 = 25;

if(num1==num2){

alert(“True”)

} else{

alert(“False”)

}

**Explicación:**

En este caso el mensaje que saldrá es “True” porque ambos poseen el mismo valor aunque no el mismo tipo.

**===**

Cuando se igualan dos datos y estos deben poseer el mismo tipo y el mismo dato.

**Ejemplo:**

var num1 = “25”;

var num2 = 25;

if(num1==num2){

alert(“True”)

} else{

alert(“False”)

}

**Explicación:**

En este caso el mensaje que saldrá es “False” porque no poseen el mismo tipo aunque ambos posean el mismo valor.

# JavaScript The Good Parts

## Capitulo 1-Good Parts

JavaScript es un lenguaje importante porque es el idioma del navegador web. JavaScript se basa en algunas ideas muy buenas y algunas muy malas.

Funciones de JavaScript son objetos de primera clase con ámbito léxico. Tiene más en común con Lisp(Es con piel de c) y Scheme que con Java.

## Capítulo 2-Grammar

**Whitespace:** Puede tomar la forma de caracteres del formato o comentarios.

**Name:** es una carta seguida opcionalmente por una o más letras, dígitos o guiones bajos. Se utilizan para declaraciones, variables, parámetros, nombres de propiedades, operadores y etiquetas.

**Number:** JavaScript tiene un solo tipo de número. A diferencia de la mayoría de otros lenguajes de programación, no hay ningún tipo de entero por separado, por lo que 1 y 1.0 son el mismo valor. El valor NaN es un valor numérico que es el resultado de una operación que no puede producir un resultado normal.

**String:** es una cadena literal que se pueden envolver en comillas simples o dobles. El \ (barra invertida) es el carácter de escape.

**Statements:** Las declaraciones tienden a ser ejecutados en orden de arriba a abajo. La secuencia de ejecución puede ser alterado por if(cambia el flujo del programa basado en el valor de la expresión), switch(realiza una rama de múltiples vías.), while(realiza un bucle simple), for(es una declaración de bucle más complicado), do(es como la instrucción while excepto que la expresión se ensaya después el bloque se ejecuta en lugar de antes), break(la salida de una sentencia de bucle o de una sentencia switch), return(hace que el pronto retorno de una función) , throw(lanza una excepción) y por la invocación de la función. Un bloque es un conjunto de sentencias envueltos entre llaves.

**Expressions:** Las expresiones más simples son un valor literal, una variable, un valor integrado, una expresión envuelto en paréntesis, una expresión precedida por un operador de prefijo. Los paréntesis se pueden utilizar para alterar la prioridad normal.Los valores producidos por typeof son number, string, boolean, undefined, function y object.

* Si el operando es null el resultado será object.
* El operador + añade o concatena. Si lo que desea es sumar asegurese de que ambos operandos sean números.
* El operador / puede producir un resultado no entero, incluso si ambos operandos son enteros.
* El operador && produce el valor de su primer operando si el primer operando es false, de lo contrario, se produce el valor del segundo operando.
* El operador || produce el valor de su primer operando si el primer operando es true, de lo contrario, se produce el valor del segundo operando.

**Literals:** Los literales de objetos son una notación conveniente para especificar los objetos nuevos. Los nombres de las propiedades se pueden especificar como nombres o como cadenas. Literales de conjunto es una notación conveniente para especificar nuevos arrays.

**Functions:** . Puede tener un nombre opcional, especificar una lista de parámetros y en su cuerpo puede incluir definiciones y declaraciones.

## Capítulo 3-Objects

Los tipos simples de JavaScript son number, string, boolean (true y false), null y undefined. Todos los demás valores son objects. Un objeto es un contenedor de propiedades, donde una propiedad tiene un nombre y un valor. Son útiles para la recopilación y organización de datos.

**Object Literals:** Un objeto literal es un par de llaves que rodean cero o más pares nombre / valor.El nombre de una propiedad puede ser cualquier cadena, incluyendo la cadena vacía. El valor de una propiedad puede ser obtenida de cualquier expresión, incluyendo otro objeto literal.

**Retrieval:** Los valores se pueden recuperar de un objeto envolviendo una expresión de cadena en un [] sufijo.

**Update:** Un valor de un objeto puede ser actualizado por asignación. Si el nombre de la propiedad ya existe en el objeto, el valor de la propiedad se sustituye.

**Reference:** Objetos se pasan alrededor por referencia.

**Prototype:** Cada objeto está vinculado a un objeto prototipo de la que puede heredar propiedades. Todos los objetos creados a partir de objetos literales están vinculados a Object.prototype, un objeto que viene de serie con JavaScript. El enlace prototipo se utiliza sólo en la recuperación.

**Reflection:** Es fácil de inspeccionar un objeto para determinar qué propiedades tiene al tratar de recuperar las propiedades y el examen de los valores obtenidos. El operador typeof puede ser muy útil para determinar el tipo de una propiedad.

**Enumeration:** La enumeración incluirá todas las propiedades, incluyendo funciones y propiedades de prototipo que podría no estar interesado en lo que es necesario para filtrar los valores que no desea. Los filtros más comunes son el método hasOwnProperty y el uso de typeof para excluir funciones

**Delete:** El operador de eliminación se puede utilizar para eliminar una propiedad de un objeto. Se eliminará una propiedad del objeto, si lo tiene.

**Global Abatement:** JavaScript hace que sea fácil de definir variables globales que pueden contener todos los activos de su aplicación. Desafortunadamente, las variables globales debilitan la resistencia de programas y deben evitarse.

## Apéndice A-Awful Parts

**Global Variables:** Una variable global es una variable que es visible en todos los ámbitos. Las variables globales pueden ser una conveniencia en programas muy pequeños, pero convertido rápidamente en difícil de manejar como los programas se hacen más grandes. El uso de variables globales degrada la fiabilidad de los programas que los utilizan.

**Scope:** Una variable declarada en un bloque es visible en todas partes en la función que contiene el bloque. Pero es mejor declarar todas las variables en la parte superior de cada función.

**Semicolon Insertion:** JavaScript tiene un mecanismo que intenta corregir los programas defectuosos por punto y coma insertar automáticamente. A veces inserta un punto y coma en lugares donde no son bienvenidos. La inserción de punto y coma lo convierte en una sentencia que devuelve undefined. No hay ninguna advertencia que punto y coma inserción causó la mala interpretación del programa.

**Reserved Words:** Las Palabras Reservadas no pueden ser utilizadas para las variables o parámetros de nombres. Cuando las palabras reservadas se utilizan como claves en objetos literales, deben ser citados. Ellas no se pueden utilizar con la notación de puntos, por lo que es necesario a veces utilizar la notación de soporte en su lugar.

**Unicode:** JavaScript fue diseñado cuando Unicode tenía como máximo 65.536 caracteres. Desde entonces ha crecido para tener una capacidad de más de 1 millón de caracteres. Unicode considera el par a ser un único carácter. JavaScript piensa la pareja es de dos personajes distintos.

**typeof:** El operador typeof devuelve una cadena que identifica el tipo de su operando, typeof no pueden distinguir entre null y objects, pero nosotros sabemos que null es false y objects es true.

**parseInt:** Es una función que convierte una cadena en un número entero. El error es cuando en la cadena aparte del numero contiene letras en este caso el parseIn será solamente el numero y no nos informara sobre el otro texto que contienen, otro de los errores es que cuando un el primer carácter de la cadena es 0, entonces la cadena se evalúa en base 8 en lugar de base 10. Por lo que el resultado de parseIn(“08”) es solamente 0. Esto provoca errores en programas con fechas y horas.

**+:** El operador + puede agregar o concatenar. Esto posee algunos problemas:

* Si alguno de los operandos es una cadena vacía, produce el otro operando convierte en una cadena.
* Si ambos operandos son números, que produce la suma.
* Si los operandos son cadenas y números convierte ambos operandos de cadenas y las concatena.

**Floating Point:** Números de punto flotante binarios son ineptos en el manejo de las fracciones decimales, por lo que 0.1 + 0.2 no es igual a 0,3. Este es el error más frecuentemente reportado en JavaScript, y es una consecuencia intencional de haber adoptado el estándar IEEE para aritmética binaria FloatingPoint (IEEE 754).

**NaN:** El valor NaN es una cantidad especial definido por IEEE 754, no es un número, a pesar de que el resultado de typeof NaN da ‘number’. El valor puede ser producido por el intento de convertir una cadena en un número cuando la cadena no está en la forma de un número . Como hemos visto, typeof no distingue entre los números y NaN, y resulta que NaN no es igual a sí mismo.JavaScript proporciona una función isNaN que puede distinguir entre los números y NaN también una función llamada isFinite es la mejor manera de determinar si un valor se puede utilizar como un número porque rechaza NaN y el infinito. Desafortunadamente, isFinite intentará convertir su operando a un número, por lo que no es una buena prueba si un valor no es en realidad un número.

**Phony Arrays:** JavaScript no tiene arrays reales. Los Arrays de JavaScript son muy fácil de usar. Sin embargo, su rendimiento puede ser considerablemente peor que los arrays reales. El operador typeof no distingue entre arrays y objetos.

**Falsy Values:** JavaScript tiene un sorprendente conjunto de valores falsos los cuales son:

|  |  |
| --- | --- |
| Valor | Tipo |
| 0 | Number |
| NaN | Number |
| “”(Comillas vacias) | String |
| False | Boolean |
| Null | Object |
| undefined | Undefined |

Estos valores todos son falsos, pero no son intercambiables.

**hasOwnProperty:** Es un método, no un operador, por lo que en cualquier objeto que pueda ser reemplazado con una función diferente o incluso un valor que no es una función.

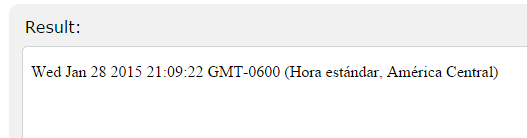
**Objects:** Objetos de JavaScript nunca son verdaderamente vacíos, ya que pueden recoger a los miembros de la cadena de prototipo

# Date()

Permite trabajar con fechas (años, meses, días, minutos, segundos, milisegundos).

**Ejemplo:**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = Date();  
</script>



**Explicación:**

En este ejemplo aparecerá la fecha exacta en el momento que el código se corra.

Fecha objetos se crean con el **new Date ()** constructor.

Hay **4 maneras** de iniciar una fecha:

new Date()  
new Date(milliseconds)  
new Date(dateString)  
new Date(year, month, day, hours, minutes, seconds, milliseconds)

Al usar new Date() crea un objeto que da la hora y la fecha actual, al crear new Date(Fecha especificada) da la fecha que dice en los paréntesis y al crear new Date(numero) da el momento cero mas el numero.

## Propiedades Date

**Constructor:** Devuelve la función que creó el prototipo del objeto Date.

**Prototype:** Le permite agregar propiedades y métodos a un objeto.

Algunos métodos son:

|  |  |
| --- | --- |
| Método | Función |
| getDate(), getDay(), getMonth(), getFullYear() | Devuelve el día del mes, el día de la semana, el mes y el año respectivamente |
| getHours(), getMinutes(), getSeconds(), getMilliseconds() | Devuelve la hora, los minutos, los segundos y los milisegundos respectivamente |
| getTime() | Devuelve el número de milisegundos desde la medianoche del 01 de enero 1970 |
| getTimezoneOffset() | Devuelve la diferencia horaria entre la hora UTC y la hora local, en minutos |

# JavaScript Tipo de Conversion

Un numero convierte a number, un string convierte a un string y un boolean conviertea booleano. Las variables de JavaScript pueden ser convertidas a una nueva variable u otro tipo de dato.

**Conversión de números a cadenas:** El método global String() convierte un número a una cadena.

**Conversión de booleanos a cadenas:** El método String() convierte de booleano a cadena.

**Conversión de fechas a cadenas:** El método String() convierte de fechas a cadena.

El método toString() hace lo mismo en todos los casos.

**Ejemplos:**

**String(​123​) ​// returns a string from a number literal 123**

**String(​false​) ​// returns "false**

**String(Date()) ​// returns Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)**

**Conversión de cadena a números:** El método Number() convierte de cadenas a números. Las cadenas vacias se convierten a 0 y las otras cadenas que no son números se convierten a NaN.

**Ejemplos:**

Number(​"3.14"​) ​// returns 3.14

Number(​" "​) ​ // returns 0

Number(​ ""​) ​ // returns 0

Number(​"99 88"​) ​// returns NaN

**Conversión de booleans a números:** El método Number() convierte de booleans a números. En donde false es 0 y true es 1.

**El operador unario +:** El operador unario + se puede utilizar para convertir una variable para un número.

**Ejemplo:**

var​ y = ​"5"​; ​// y is a string

var​ x = + y; ​// x is a number

**Conversión de fechas a números:** El método Number() convierte de fechas a números.

**Ejemplo:**

d = ​new​ Date();

Number(d) ​// returns 1404568027739

El método de la fecha getTime () hace lo mismo.

d = ​new​ Date();

d.getTime() ​// returns 1404568027739

# La Propiedad Constructor

Devuelve la función constructora para todas las variables de JavaScript.

**Ejemplo:**

"John"​.constructor ​ // Returns function String() { [native code] }

(​ 3.14​).constructor ​ // Returns function Number() { [native code] }

false.constructor ​ // Returns function Boolean() { [native code] }

[​ 1​ ,​ 2​ ,​ 3​ ,​ 4​ ].constructor ​ // Returns function Array() { [native code] }

{name:​'John'​, age:34}.constructor ​// Returns function Object() { [native code] }

new​ Date().constructor ​// Returns function Date() { [native code] }

function​ () {}.constructor ​ // Returns function Function(){ [native code] }

# JavaScript Izar

Izar es el comportamiento predeterminado de JavaScript de mover todas las declaraciones de la parte superior del ámbito actual a la parte superior de la secuencia de comandos actual o la función actual. En JavaScript, una variable puede ser declarada después de que se ha utilizado. En otras palabras; una variable puede ser utilizada antes de que haya sido declarada.

**Ejemplo:**

x = ​ 5​ ; // Assign 5 to x

elem = document.getElementById(​"demo"​); ​// Find an element

elem.innerHTML = x; ​// Display x in the element

var​ x; ​// Declare x

JavaScript sólo iza declaraciones, no inicializaciones.

**Ejemplo:**

var​ x = ​ 5​ ; ​// Initialize x

elem = document.getElementById(​"demo"​); ​// Find an element

elem.innerHTML = x + ​" "​ + y; ​// Display x and y

var​ y = ​ 7​ ; ​// Initialize y

**Explicación:**

En esta caso y no tendrá un valor definido.

# El Use Strict

La directiva "use strict" es nuevo en JavaScript 1.8.5 (ECMAScript versión 5). No es una declaración, sino una expresión literal, ignorado por las versiones anteriores de JavaScript. El propósito de "use strict" es para indicar que el código debe ser ejecutado en el "modo estricto". Con el modo estricto no se puede, por ejemplo, utilizar variables no declaradas.

**Declarando el Modo Estricto:**

El modo estricto se declara mediante la adición de "uso estricto"; para el comienzo de un JavaScript archivo, o una función de JavaScript. Declarado desde el inicio del archivo tiene alcance global y declarado desde una función tiene alcance local.

**Declaración global:**

"use strict"​ ;

x = ​3.14​;

myFunction();

function​ myFunction() {

x = ​3.14​ ;

}

**Declaración local:**

x = ​3.14​;

myFunction();

function​ myFunction() {

​"use strict"​ ;

x = ​3.14​ ;

}

# Errores Comunes en JavaScript

**Accidentalmente Usando el operador de  asignación:** Uno de los problemas es si un programador utiliza accidentalmente una misiónoperador (=), en lugar de un operador de comparación (==) en una sentencia if.

**Ejemplos:**

var​ x = ​ 0​ ;

if​ (x == ​ 10)

Esto da como resultado false.

var​ x = ​ 0​ ;

if​ (x =​ 10)

Esto da como resultado true.

var​ x = ​ 0​ ;

if​ (x = 0)

Esto da como resultado false.

**Esperando Libremente Comparación:** Es un error común olvidar que los interruptores declaraciones utilizan comparación estricta.

**Ejemplos:**

var​ x = ​ 10​ ;

switch​(x) {

​case​ ​ 10​: alert(​"Hello"​ );

}

En este caso se mostrara el alert.

var​ x = ​ 10​ ;

switch​(x) {

​case​ ​ "10"​: alert(​"Hello"​ );

}

En este caso no se mostrara el alert.

**Confundir adición y concatenación:** Otro de los errores comunes es adición y concatenación. Adición es cuando se suman dos números y concatenación es cuando dos cadenas se concatenan. Los errores que se cometen es cuando se intenta sumar un número con otro número que realmente es un string.

**Ejemplo:**

var​ x = ​ 10​ + ​ 5​ ; ​// the result in x is 15

En este caso la suma es correcta ya que ambos son numeros

var​ x = ​ 10​ + ​"5"​; ​// the result in x is "105

En cambio en este caso lo que hace es concatenar el 10 y la cadena que contiene el numero 5.

**Incomprensión** **Flota:** Todos los números en JavaScript se almacenan como 64-bits de números de punto flotante (flotadores).

**Ejemplo:**

var​ x = ​0.1​ ;

var​ y = ​0.2​ ;

var​ z = x + y ​// the result in z will not be 0.3

if​ (z == ​0.3​) ​// El resultado del if será false.

**Romper una cadena JavaScript:** JavaScript se permitirá romper una declaración en dos líneas pero, rompiendo un comunicado en el medio de una cadena no funcionará. Si es necesario romper una cadena se debe utilizar la barra invertida “\”.

**Ejemplo:**

var​ x =

"Hello World!";

var​ x = ​"Hello

World!";

var​ x = ​"Hello \

World!";

**Extraviar punto y coma:** Debido a un punto y coma fuera de lugar, este bloque de código se ejecutará independientemente del valor de x:

**Ejemplo:**

if​ (x == ​ 19​ ); {

​// code block

}

**Rompiendo una sentencia return:** Se trata de un comportamiento por defecto JavaScript para cerrar un comunicado automáticamente al final de una línea.

**Ejemplo:**

function​ myFunction(a) {

​var​ power = ​ 10

​return​ a \* power

}

function​ myFunction(a) {

​var​ power = ​ 10​ ;

​ return​ a \* power;

}

En ambos Ejemplos se retornara el mismo valor.

En caso de que la sentencia de return se rompiera el resultado será undefined.

function​ myFunction(a) {

​var power = ​ 10​ ;

​return

a \* power;

}

JavaScript cree que lo que quisiste hacer es colocar después del return un punto y coma (;).

**Acceso a arrays con índices con nombre:** JavaScript no no admite matrices con índices con nombre. En JavaScript, los arrays utilizan índices numerados. Al llamar a un índice de un array con nombre JavaScript redefinirá el array como un objeto estándar y esto traerá resultados undefined e incorrectos.

**Ejemplo:**

var​ person = [];

person[​"firstName"​] = ​"John"​ ;

person[​"lastName"​] = ​"Doe"​ ;

person[​"age"​] = ​ 46​ ;

var​ x = person.length; ​// person.length will return 0

var​ y = person[​ 0​ ]; ​// person[0] will return undefined

**Finalización de una definición del array con una coma:** Algunos motores de JSON y JavaScript fracasarán, o comportarse de forma inesperada

**Incorrecto:**

points = [​ 40​ , ​100​ , ​ 1​ , ​ 5​ , ​ 25​ , ​ 10​,];

**Correcto:**

points = [​ 40​ , ​100​ , ​ 1​ , ​ 5​ , ​ 25​ , ​ 10​ ];

**Poner fin a una definición de objeto con una coma:**

**Incorrecto:**

person = {firstName:​"John"​, lastName:​"Doe"​, age:​ 46​ ,}

**Correcto:**

person = {firstName:​"John"​, lastName:​"Doe"​, age:​ 46​}

**Indefinido no es nulo:** Con JavaScript, nula es para objetos, indefinido es para variables, propiedades y métodos. Para ser nula, un objeto tiene que ser definido, de lo contrario será indefinido.

**Incorrecto:**

if​ (myObj !== ​null​ && ​typeof​ myObj !== ​"undefined"​

**Correcto:**

if​ (​typeof​ myObj !== ​"undefined"​ && myObj !== ​null

**Esperando Ámbito de nivel de bloque:** JavaScript no crear un nuevo ámbito para cada bloque de código. Es cierto en muchos lenguajes de programación, pero no es cierto en JavaScript.

**Ejemplo:**

for​ (​var​ i = ​ 0​ ; i < ​ 10​; i++) {

​// some code

}

return​ i;

Esto devolverá undefined.

# parseInt:

La función parseInt () analiza una cadena y devuelve un numero entero. El parámetro radix se utiliza para especificar qué sistema de numeración se va a utilizar.

**Ejemplo:**

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;

Los resultados de estos ejemplos son:

10  
10  
10  
34  
60  
40  
NaN  
  
10  
10  
8  
16  
16

**Explicacion:**

En este ejemplo lo que pasa es que cada variable contiene un parseInt y en los paréntesis introduce un string, al imprimir esto el resultado de todos serán números.

# parseFloat

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, se analiza la cadena hasta que llega al final del número, y devuelve el número como un número, no como una cadena.

**Ejemplo:**

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>";

El reultado será:

10  
10  
10.33  
34  
60  
40  
NaN

**Explicación:**

El resultado es un número pero este si tiene decimales los muestra.

# indexOf

El método indexOf () devuelve la posición de la primera aparición de un valor especificado en una cadena. Este método devuelve -1 si el valor de búsqueda nunca ocurre.

**Ejemplo:**

var str = "Hello world, welcome to the universe.";  
var n = str.indexOf("welcome");

El resultado de n será:

13

**Explicacion:**

El ejemplo de vuelve la posición de la primera letra de la palabra dentro del paréntesis, es decir, en indexOf(“welcome”) devuelve en qué posición de la oración anterior (var str) de la letra “w”.