Kenia Montano Granados

Resumen JavaScript

Contenido

¿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.

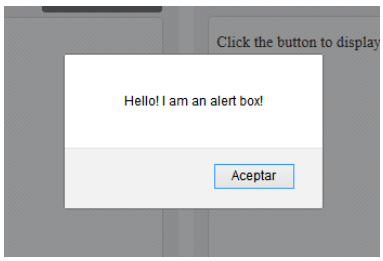
alert()

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:

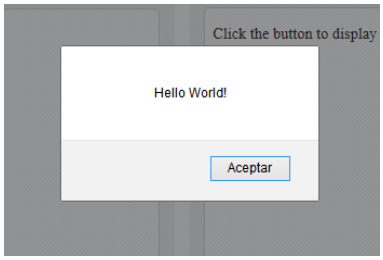
Es muy sencillo solamente aparece el texto que esta dentro del paréntesis.

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.

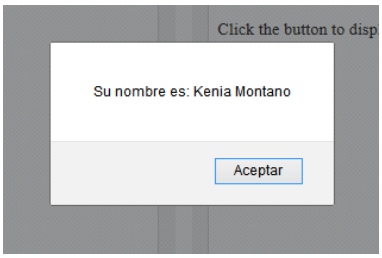
También podemos mezclar ambos ejemplos.

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()

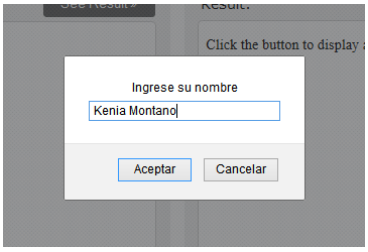
Este muestra un mensaje pidiendo el usuario que ingrese un dato (nombre, edad, nacionalidad, color, etc.) al ingresar lo que se pide, saldrá un mensaje, este mensaje dependerá de la información suministrada.

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:

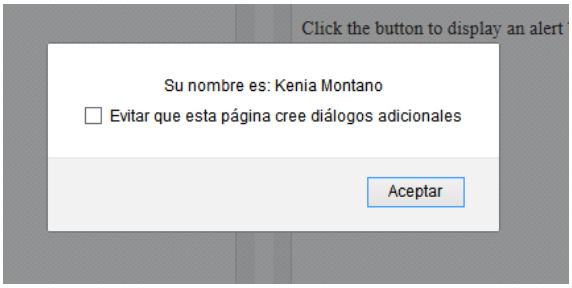
```
var str = prompt(“Ingrese su nombre”);
```

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).

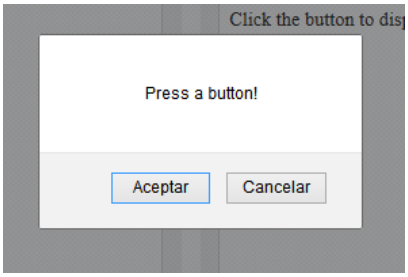
confirm()

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

Capítulo 4-Funciones

Las funciones son la unidad modular fundamental de JavaScript. Se utilizan para la reutilización de código, ocultación de información, y la composición.

Objetos de Función: Las funciones en JavaScript son objetos. Cada función también se crea con dos propiedades adicionales ocultas: contexto de la función y el código que implementa el comportamiento de la función. Las funciones pueden ser almacenadas en las variables, objetos y arrays. Se pueden pasar como argumentos a funciones, y pueden ser devueltos por funciones.

Función Literal: Un literal de función tiene cuatro partes:

1. La primera parte es la palabra reservada function.
2. El nombre de la función, si no se le diera nombre, la función seria anónima.
3. Los parámetros de la función, envueltos en paréntesis.
4. La cuarta parte es un conjunto de sentencias envueltos entre llaves.

Invocación: Invocar una función suspende la ejecución de la función actual, pasando de control y parámetros para la nueva función. Cada función recibe dos parámetros adicionales: this y arguments.

This: es muy importante en la programación orientada a objetos, y su valor se determina por el patrón de invocación. Existen cuatro patrones de invocación los cuales son:

- El patrón de invocación de métodos.
- El patrón de invocación de la función.
- El patrón de invocación del constructor.
- El patrón de invocación se aplican.

Arguments: los argumentos se le asignarán nombres de los parámetros de la función. No hay error de ejecución cuando el número de argumentos y el número de parámetros no coinciden. Si hay demasiados valores de los argumentos, se ignorarán los valores de los argumentos adicionales. Si hay muy pocos valores de los argumentos, el valor indefinido será sustituido por los valores perdidos.

El patrón de Invocación de Método: Cuando una función se almacena como una propiedad de un objeto, lo llamamos un método. Un método puede usar esto para acceder al objeto para que pueda recuperar los valores del objeto o modificar el objeto. La unión de este con el objeto pasa al momento de la invocación.

El patrón de Invocación de Función: Cuando una función no es la propiedad de un objeto, entonces se invoca como una función Cuando se invoca una función con este patrón, esto se enlaza con el objeto global.

El Patrón de Invocación del Constructor: Funciones que están destinados a ser utilizados con el nuevo prefijo se llaman constructores. Por convención, se mantienen en las variables con un nombre en mayúsculas. Si un constructor se llama sin el nuevo prefijo, cosas muy malas pueden suceder sin una advertencia de tiempo de compilación o en tiempo de ejecución, por lo que la convención de la capitalización es realmente importante.

El patrón de invocación de Aplicar: El método de aplicación nos permite construir una serie de argumentos a utilizar para invocar una función. También nos permite elegir el valor de este. El método de aplicación tiene dos parámetros. El primero es el valor que se debe obligado a ello. El segundo es una serie de parámetros.

Argumentos: Un parámetro bono que está disponible para funciones cuando se invocan es el conjunto argumentos. Se da el acceso a las funciones de todos los argumentos que se suministraron con la invocación, incluyendo el exceso de argumentos que no fueron asignados a los parámetros.

Return: Cuando se ejecuta el retorno, la función devuelve inmediatamente sin ejecutar las sentencias restantes. Una función siempre devuelve un valor. Si no se especifica el valor de retorno, a continuación, se devuelve undefined.

Excepciones: Las excepciones son inusuales percances que interfieren con el flujo normal de un programa. La declaración de banda interrumpe la ejecución de la función. Debe administrarse un objeto de excepción que contiene una propiedad nombre que identifica el tipo de excepción, y una propiedad de mensaje descriptivo. También puede añadir otras propiedades. El objeto de excepción será entregado a la cláusula catch de una sentencia try.

Aumentar tipos: Al aumentar Function.prototype con un método, ya no tenemos que escribir el nombre de la propiedad prototipo. Esa parte de la fealdad ahora se puede esconder. Mediante el aumento de Function.prototype, podemos hacer un método disponible para todas las funciones:

```
Function.prototype.method = function (name, func) {
  this.prototype[name] = func;
  return this;
};
```

Al aumentar los tipos básicos, podemos hacer mejoras significativas a la expresividad del lenguaje. Debido a la naturaleza dinámica de las herencias de prototipos de JavaScript, todos los valores están dotados de inmediato con los nuevos métodos, incluso los valores que se crearon antes de la creación de los métodos.

La recursividad: Una función recursiva es una función que llama a sí misma, ya sea directa o indirectamente. En general, una función recursiva se llama a resolver sus subproblemas.

Alcance: Alcance en un lenguaje de programación controla la visibilidad y tiempos de vida de las variables y parámetros. Este es un servicio importante para el programador, ya que reduce las colisiones de nombres y proporciona gestión automática de memoria. Los parámetros y las variables definidas en una función no son visibles fuera de la función, y que una variable definida en cualquier lugar dentro de una función es visible en todas partes dentro de la función. Lo mejor es declarar todas las variables utilizadas en una función en la parte superior del cuerpo de la función.

Cierre: La buena noticia sobre el alcance es que las funciones internas tienen acceso a los parámetros y variables de las funciones que están definidas dentro (con la excepción de this y arguments). La función que tiene acceso al contexto en el que se creó a esto se le denomina cierre.

Devoluciones de Llamadas: Las funciones pueden hacer que sea más fácil lidiar con eventos discontinuos. Si la red o el servidor son lentos, la degradación en la capacidad de respuesta será inaceptable. Un mejor enfoque es hacer una solicitud asincrónica, que proporciona una función de devolución de llamada que se invoca cuando se recibió la respuesta del servidor. Una función asíncrona devuelve inmediatamente, por lo que el cliente no está bloqueado.

Módulo: Podemos utilizar las funciones y cierre para hacer módulos. Un módulo es una función u objeto que presenta una interfaz pero que esconde su estado y la aplicación. Mediante el uso de funciones para producir módulos, podemos eliminar casi por completo el uso de variables globales, mitigando de esta manera una de las peores características de JavaScript. El patrón módulo se aprovecha de ámbito de la función y el cierre de crear relaciones que son vinculantes y privadas. El uso del patrón de módulo puede eliminar el uso de variables globales. También se puede utilizar para producir objetos que son seguros.

Cascada: En una cascada, podemos llamar a muchos métodos en el mismo objeto en secuencia en una sola sentencia. En cascada puede producir interfaces que son muy expresivos. Se puede ayudar a controlar la tendencia a hacer interfaces que tratan de hacer demasiadas cosas a la vez.

Memoization: Las funciones se pueden usar objetos para recordar los resultados de las operaciones anteriores, por lo que es posible evitar trabajo innecesario. Esta optimización se llama memoization. Objetos y arrays de JavaScript son muy convenientes para esto.

Capítulo 5- Inheritance

En las lenguas clásicas (como Java), la herencia proporciona dos servicios útiles. . En primer lugar, es una forma de reutilización de código. Si una nueva clase es principalmente similar a una clase existente, sólo tiene que especificar las diferencias. La otra ventaja de la herencia clásica es que incluye la especificación de un sistema de tipos. Esto libera sobre todo al programador de tener que escribir las operaciones de fundición explícitas, que es una cosa muy buena porque cuando se lanza, se pierden los beneficios de seguridad de un sistema de tipos. . En las lenguas clásicas, los objetos son instancias de clases, y una clase puede heredar de otra clase. JavaScript es un lenguaje de prototipos, lo que significa que los objetos heredan directamente de otros objetos.

Pseudoclassical: El nuevo objeto de función se da una propiedad prototipo cuyo valor es un objeto que contiene una propiedad constructor cuyo valor es el nuevo objeto de función. El objeto prototipo es el lugar donde los rasgos hereditarios son a depositar. Cada función obtiene un objeto prototipo porque el lenguaje no proporciona una manera de determinar qué funciones están destinados a ser utilizados como constructores. . La forma pseudoclassical puede proporcionar comodidad a los programadores que no están familiarizados con el lenguaje Java, pero también oculta la verdadera naturaleza de la lengua. La notación de inspiración clásica puede inducir a los programadores a componer jerarquías que son innecesariamente profundos y complicados.

Object Specifiers: A veces sucede que un constructor se le da un número muy grande de parámetros. Esto puede ser problemático porque puede ser muy difícil de recordar el orden de los argumentos. En tales casos, puede ser mucho más amigable si escribimos el constructor para aceptar un único objeto especificador lugar.

Ejemplo:

```
var myObject = maker(f, l, m, c, s);
```

Podemos escribir:

```
var myObject = maker({
   first: f,
   last: l,
   state: s,
   city: c
});
```

Los argumentos ahora se pueden enumerar en cualquier orden, los argumentos pueden ser dejados de lado si el constructor es inteligente acerca de incumplimientos, y el código es mucho más fácil de leer. Esto puede tener un beneficio secundario al trabajar con JSON.

Prototypal: Herencias de prototipos es conceptualmente más simple que la herencia clásica: un nuevo objeto puede heredar las propiedades de un objeto antiguo. Se empieza por hacer un objeto útil. A continuación, puede hacer muchos más objetos que son como ese. Mediante la personalización de un nuevo objeto, especificamos las diferencias con el objeto sobre el que se basa.

Functional: Una de las debilidades de los patrones de herencia que hemos visto hasta ahora es que no recibimos ninguna privacidad. Todas las propiedades de un objeto son visibles. Llegamos sin variables privadas y no hay métodos privados. El patrón funcional tiene una gran flexibilidad. Requiere menos esfuerzo que el patrón pseudoclassical, y nos da una mejor encapsulación y ocultación de información y acceso a métodos de super. Si creamos un objeto en el estilo funcional, y si todos los métodos del objeto no hacen uso de esto o aquello, entonces el objeto es durable. Un objeto durable es simplemente una colección de funciones que actúan como capacidades.

Parts: Podemos componer objetos de conjuntos de piezas. Por ejemplo, podemos hacer una función que puede agregar características simples de procesamiento de eventos a cualquier objeto. Añade una sobre el método, un método de fuego, y un registro de evento privado.

Apendice B-Bad Parts

==: JavaScript tiene dos grupos de operadores de igualdad: **===** y **!==** , y sus gemelos malvados **==** y **! =**. Si los dos operandos son del mismo tipo y tienen el mismo valor, entonces === produce verdadero y !== Produce false. Los gemelos malvados hacen lo correcto cuando los operandos son del mismo tipo, pero si son de diferentes tipos, en su intento de coaccionar a los valores. Lo mejor es nunca utilizar los gemelos malvados.

with Statement: JavaScript tiene una declaración que fue con la intención de proporcionar un atajo al acceder a las propiedades de un objeto. Por desgracia, sus resultados pueden ser a veces impredecible, por lo que deben evitarse.

Ejemplo:

```
with (obj) {
   a = b;
 }
```

eval: La función eval pasa una cadena al compilador JavaScript y ejecuta el resultado. Es el hecho más mal utilizado de JavaScript. Algunos problemas son:

- Esta forma será mucho más lento, ya que necesita para ejecutar el compilador sólo para ejecutar una sentencia de asignación trivial.
- También frustra JSLint, por lo que la capacidad de la herramienta para detectar problemas se reduce significativamente.
- La función eval también pone en peligro la seguridad de su aplicación, ya que otorga demasiada autoridad al texto eval'd.
- Y pone en peligro el rendimiento de la lengua como un todo de la misma manera que la sentencia with hace.

La forma de argumento de cadena también debe ser evitado.

continue Statement: La sentencia continue salta a la parte superior del bucle. Nunca se ha visto un trozo de código que no fue mejorada por la refactorización para eliminar la sentencia continue.

switch Fall Through: La sentencia switch fue modelado después del FORTRAN IV computarizada ir a la declaración. Cada caso entra a través en el siguiente caso a menos que interrumpes explícitamente el flujo.

Block-less Statements: El formulario de declaración solo es otra molestia atractiva. Ofrece la ventaja de ahorrar dos personajes, una ventaja dudosa. Se oscurece la estructura del programa para que los manipuladores posteriores del código pueden insertar fácilmente los insectos. Los programas que aparecen para hacer una cosa pero hacen otra realidad es mucho más difícil hacerlo bien. Un uso disciplinado y consistente de bloques hace que sea más fácil de hacer las cosas bien.

++ --: Los operadores de incremento y decremento permiten escribir en un estilo muy conciso. También fomentan un estilo de programación que, como resulta, es imprudente. La mayoría de los errores de desbordamiento de buffer que crearon vulnerabilidades de seguridad terribles se debieron a codificar como este.

Bitwise Operators: JavaScript tiene el mismo conjunto de operadores de bits como Java:

&	and
|	or
^	xor
~	not
>>	signed right shift
>>>	unsigned right shift

<<	left shift

JavaScript no tiene números enteros. Sólo tiene los números de coma flotante de doble precisión. Por lo tanto, los operadores de bits convierten sus operandos numéricos en enteros, hacen sus negocios, y luego convertirlos de nuevo. En la mayoría de idiomas, estos operadores están muy cerca del hardware y muy rápido. En JavaScript, están muy lejos de ser el hardware y muy lento.

The function Statement vs the function Expression: JavaScript tiene una sentencia de función, así como una expresión de función. Esto es confuso, ya que pueden tener el mismo aspecto. Una declaración de la función es la abreviatura de una sentencia var con un valor de la función. Es importante entender que las funciones son valores. La primera cosa en una declaración no puede ser una expresión de función porque la gramática oficial asume que una declaración que comienza con la palabra función es una sentencia de función. La solución consiste en envolver la expresión de función entre paréntesis.

Typed Wrappers: JavaScript tiene un conjunto de envolturas mecanografiadas. Es mejor evitar número booleano o una nueva cadena. También evite nuevo Objeto y new Array. Use {} y [] en lugar.

new: Nuevo operador de JavaScript crea un nuevo objeto que hereda de miembro prototipo del operando, y luego llama el operando, la unión del nuevo objeto a esto. Lo mejor afrontamiento es no utilizar new en absoluto.

void: En muchos idiomas, nula es un tipo que no tiene valores. En JavaScript, nula es un operador que toma un operando y devuelve indefinido. Esto no es útil, y es muy confuso. Evite vacío.

Capítulo 6-Array

JavaScript proporciona un objeto que tiene algunas características en arreglos similares. Convierte subíndices de matriz en cadenas que se utilizan para hacer propiedades. Es significativamente más lento que una matriz real, pero puede ser más conveniente de usar. Recuperación y actualización de las propiedades funcionan igual que con los objetos, excepto que hay un truco especial con nombres de propiedad enteros. Matrices tienen su propio formato literal.

Array Literal: Los arreglos literales proporcionan una notación muy conveniente para la creación de nuevos valores de la matriz. Un literal de matriz es un par de corchetes cero o más valores separados por comas. El primer valor tendrá el nombre de la propiedad '0', el segundo valor tendrá el nombre de la propiedad "1", y así sucesivamente.

En la mayoría de los idiomas, los elementos de una matriz están todos obligados a ser del mismo tipo. JavaScript permite una matriz para contener cualquier mezcla de valores.

Length: Cada matriz tiene una propiedad length. A diferencia de otros idiomas, longitud de la matriz de JavaScript no es un límite superior. Si almacena un elemento con un subíndice que es mayor que o igual a la longitud actual, la longitud aumentará para contener el nuevo elemento. Un

nuevo elemento se puede añadir al final de una matriz mediante la asignación a la longitud actual de la matriz pero es más conveniente utilizar el método push para lograr la misma cosa.

Delete: Desde matrices de JavaScript son realmente objetos, el operador delete puede utilizarse para eliminar elementos de un array. Por desgracia, que deja un agujero en la matriz. Afortunadamente, las matrices de JavaScript tienen el método splice. El primer argumento es un ordinal de la matriz. El segundo argumento es el número de elementos que desea eliminar.

Delete:

```
delete numbers[2];
// numbers is ['zero', 'one', undefined, 'shi', 'go']
```

Splice:

```
numbers.splice(2, 1);
 // numbers is ['zero', 'one', 'shi', 'go']
```

Enumeration: En el que hace ninguna garantía sobre el orden de las propiedades, y la mayoría de las aplicaciones de arreglo esperan que los elementos que se producen en orden numérico. Afortunadamente, la convencional para la declaración evita estos problemas. De JavaScript para la declaración es similar a la de la mayoría de los lenguajes de programación como C. Está controlado por tres cláusulas de la primera inicializa el bucle, el segundo es la condición rato, y el tercero hace la subasta.

Confusion: Un error común en los programas JavaScript es utilizar un objeto cuando se requiere una matriz o una matriz cuando se requiere un objeto. La regla es simple: cuando los nombres de propiedad son pequeños números enteros consecutivos, se debe utilizar una matriz. De lo contrario, utilice un objeto.

Methods: JavaScript proporciona un conjunto de métodos para actuar sobre arrays. Los métodos son funciones almacenadas en Array.prototype. dos de esos métodos son reduce(add,x) y reduce(mult, x) que sirven para sumar y multiplicar, respectivamente.

Dimensions: Matrices de JavaScript por lo general no se inicializan. Si usted pide una nueva matriz con [], que estará vacío. Si va a implementar algoritmos que asumen que todos los elementos se inician con un valor conocido (como 0), entonces usted debe preparar la matriz a ti mismo. JavaScript debe haber proporcionado algún tipo de un método Array.dim para hacer esto, pero podemos corregir fácilmente este descuido.

Capítulo 8-Methods

JavaScript incluye un pequeño conjunto de métodos estándar que están disponibles en los tipos estándar.

Array

array.concat(item…): El método concat produce una nueva matriz que contiene una copia superficial de esta matriz con los elementos añadidos al final.

array.join(separator): El método join hace una cadena de una matriz. Para ello, hacer una cadena de cada uno de los elementos de la matriz y, a continuación, concatenar todos juntos con un separador entre ellos.

*array.pop():*El método pop elimina y devuelve el último elemento de esta matriz. Si la matriz está vacía, devuelve undefined.

array.push(item…): El método push añade elementos al final de una matriz.

*array.reverse():*El método modifica la matriz inversa invirtiendo el orden de los elementos.

array.shift(): El método de cambio elimina el primer elemento de una matriz y devuelve.

array.slice(start,end): El método slice hace una copia superficial de una porción de una matriz. El primer elemento copiado será array [Inicio]. Se detendrá antes de copiar array [final]. El parámetro final es opcional y el valor predeterminado es Array.length.

array.sort(comparefn): El método sort ordena el contenido de una matriz en su lugar. Función de comparación por defecto de JavaScript asume que los elementos a ser ordenados son cadenas. No es lo suficientemente inteligente como para probar el tipo de los elementos antes de compararlos, lo que convierte los números a cadenas como las compara, asegurando un resultado sorprendentemente incorrecta.

array.splice(start,deleteCount,item…): El método splice elimina elementos de una matriz, reemplazándolos con nuevos elementos. El parámetro de start es el número de una posición dentro de la matriz. El parámetro deleteCount es el número de elementos a eliminar a partir de esa posición. Si hay parámetros adicionales, esos elementos se insertarán en la posición.

array.unshift(item…): El método unshift es como el método push excepto que empuja a los elementos al frente de esta matriz en lugar de al final.

Function:

function.apply(thisArg,argArray): El método de aplicación invoca una función, pasando el objeto que se une a esto y un conjunto opcional de argumentos.

Numbers:

number.toExponential(fractionDigits: El método toExponential convierte este número en una cadena en forma exponencial. El parámetro fractionDigits opcional controla el número de decimales.

number.toFixed(fractionDigits): El método toFixed convierte este número en una cadena en forma decimal. El parámetro fractionDigits opcional controla el número de decimales.

number.toPrecision(precision): El método toPrecision convierte este número en una cadena en forma decimal. El parámetro de precisión opcional controla el número de dígitos de precisión.

number.toString(radix): El método toString convierte este número en una cadena. El parámetro opcional radix controla radix, o base. El valor predeterminado es radix base 10.

Object:

object.hasOwnProperty(name): El método hasOwnProperty devuelve true si el objeto contiene una propiedad que tiene el nombre.

RegExp:

regexp.exec(string): El método exec es la más potente (y más lenta) de los métodos que utilizan expresiones regulares. Si coincide con éxito la expresión regular y la cadena, devuelve una matriz. El elemento 0 de la matriz contendrá la subcadena que hacía juego con la expresión regular. El elemento 1 es el texto capturado por el grupo 1, el elemento 2 es el texto capturado por el grupo 2, y así sucesivamente.

regexp.test(string): El método de prueba es el más sencillo (y más rápido) de los métodos que utilizan expresiones regulares. Si la expresión regular coincide con la cadena, devuelve true; de lo contrario, devuelve false.

String:

string.charAt(pos): El método charAt devuelve el carácter en la posición pos en esta cadena. Si pos es menor que cero o mayor que o igual a String.length, devuelve la cadena vacía. JavaScript no tiene un tipo de carácter. El resultado de este método es una cadena.

string.charCodeAt(pos): El método charCodeAt es el mismo que charAt excepto que en lugar de devolver una cadena, se devuelve una representación entera del valor de punto de código del carácter en la posición pos de esa cadena. Si pos es menor que cero o mayor que o igual a string.length, devuelve NaN.

string.concat(string…): El método concat hace una nueva cadena concatenando otras cadenas juntas.

string.indexOf(searchString,position): El método indexOf busca una searchstring dentro de una cadena. Si se encuentra, devuelve la posición del primer carácter emparejado; de lo contrario, devuelve -1.

string.lastIndexOf(searchString,position): El método lastIndexOf es como el método indexOf, excepto que busca desde el extremo de la cadena en lugar de la parte delantera.

string.localeCompare(that): El método localCompare compara dos cadenas. No se especifican las reglas de cómo se comparan las cuerdas. Si esta cadena es menor que la cadena, el resultado es negativo. Si son iguales, el resultado es cero.

string.match(regexp): El método partido coincide con una cadena y una expresión regular. Si no hay ningún indicador g , entonces el resultado de la llamada String.match (regexp) es el mismo que llamar regexp.exec (string) . Sin embargo, si la expresión regular tiene el indicador g , entonces se produce una matriz de todos los partidos, pero excluye a los grupos de captura.

string.replace(searchValue,replaceValue): El método replace hace una búsqueda , reemplaza la operación en esta cadena y produce una nueva cadena. El argumento searchValue puede ser una cadena o un objeto de expresión regular. Si searchValue es una expresión regular y si tiene el indicador g, entonces reemplazará todas las apariciones.

string.search(regexp): El método de búsqueda es como el método indexOf, excepto que toma un objeto de expresión regular en lugar de una cadena.

string.slice(start,end): El método slice hace una nueva cadena copiando una parte de otra cadena. Si el parámetro de inicio es negativo, se añade string.length a ella. El parámetro final es opcional, y su valor por defecto es string.length. Si el parámetro final es negativo, entonces string.length se añade a la misma.

string.split(separator,limit): El método split crea una matriz de cadenas mediante el fraccionamiento de esta cadena en trozos. El parámetro de límite opcional puede limitar el número de piezas que se dividen. El parámetro separador puede ser una cadena o una expresión regular.

string.substring(start,end): El método subcadena es el mismo que el método slice excepto que no maneja el ajuste de los parámetros negativos.

*string.toLocaleLowerCase():*El método toLocaleLowerCase produce una nueva cadena que se hizo mediante la conversión de esta cadena a minúsculas utilizando las reglas para la configuración regional.

*string.toLocaleUpperCase():*El método toLocaleUpperCase produce una nueva cadena que se hizo mediante la conversión de esta cadena a mayúsculas utilizando las reglas para la configuración regional.

*string.toLowerCase():*El método toLowerCase produce una nueva cadena que se hizo mediante la conversión de esta cadena a minúsculas.

*string.toUpperCase():*El método toUpperCase produce una nueva cadena que se hizo mediante la conversión de esta cadena a mayúsculas.

String.fromCharCode(char…): La función String.fromCharCode produce una cadena a partir de una serie de números.

Capítulo 9-Style

Los programas se componen de un gran número de piezas, expresados como funciones, declaraciones y expresiones que se organizan en secuencias que deben estar prácticamente libres de error. Los buenos programas tienen una estructura que anticipa, pero no está demasiado agobiado por caso las posibles modificaciones que serán necesarias en el futuro. Los buenos programas también tienen una presentación clara. Si un programa se expresa así, entonces tenemos la mejor oportunidad de ser capaz de entenderlo de modo que pueda ser modificado o reparado con éxito.

JavaScript contiene un amplio conjunto de características débiles o problemáticos que pueden socavar nuestros intentos de escribir buenos programas. Obviamente debemos evitar peores características de JavaScript. Sorprendentemente, quizás, también debemos evitar las características que a menudo son útiles pero en ocasiones peligrosos. Si un programa es capaz de comunicar con claridad su estructura y características, es menos probable que se rompa cuando se modifica en el futuro nunca demasiado lejano. Al escribir en un estilo claro y consistente, sus programas son más fáciles de leer.

El código que parece significar una cosa, pero en realidad significa otra es probable que cause errores. Un par de llaves es la protección realmente barato contra los insectos que pueden ser caros de encontrar. Hay que poner una atención especial a las características que a veces son útiles, pero en ocasiones peligrosas. Esas son las peores partes, ya que es difícil decir si se están utilizando correctamente. Es un lugar donde los insectos se esconden. La calidad no era una preocupación lenta en el diseño, implementación, o estandarización de JavaScript.

Capítulo 10- Beautiful Features

Simplificado JavaScript es sólo las cosas buenas, incluyendo:

Funciones como la primera clase

Objetos Funciones en Simplificado JavaScript son lambdas con ámbito léxico.

Los objetos dinámicos con objetos herencias de prototipos son de clase gratis.

Podemos agregar un nuevo miembro a cualquier objeto de cesión ordinaria. Un objeto puede heredar los miembros de otro objeto.

Los literales de objetos y literales de matriz

Esta es una notación muy conveniente para la creación de nuevos objetos y arrays. Literales de JavaScript fue la inspiración para el formato de intercambio de datos JSON.

Simplificado JavaScript no es estrictamente un subconjunto. Usted puede usar una palabra para uno o el otro, y el programador tiene que elegir. Eso hace que una lengua fácil de aprender, ya que no tienen que ser conscientes de las características que no usa. Y hace que el idioma más fácil de extender, porque no es necesario reservar más palabras para añadir nuevas funciones.

El subconjunto adicional, sirve para mostrar cómo tomar un lenguaje de programación existente y hacer mejoras significativas a ella por no hacer cambios, excepto para excluir las características de

bajo valor. En los sistemas de software, hay un costo de almacenamiento, que se estaba convirtiendo en insignificantes, pero en las aplicaciones móviles se está convirtiendo en importante otra vez. Características tienen un costo documentación. Cada característica añade páginas al manual, aumentando los costos de formación. Características que ofrecen valor a una minoría de usuarios e imponen un costo a todos los usuarios. Por lo tanto, en el diseño de productos y lenguajes de programación, queremos conseguir la característica del núcleo buena parte derecha porque es donde creamos la mayor parte del valor.

Sería bueno si los productos y lenguajes de programación fueron diseñadas para tener sólo partes buenas.

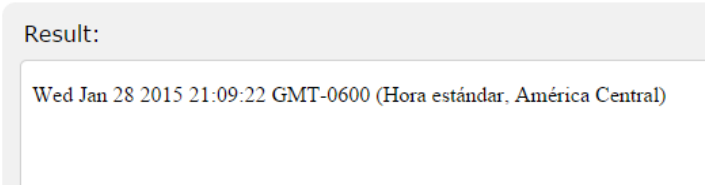
Date()

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:	**Correcto:**
points = [40 , 100 , 1 , 5 , 25 , 10,];	points = [40 , 100 , 1 , 5 , 25 , 10];

Poner fin a una definición de objeto con una coma:

Incorrecto:	**Correcto:**
person = {firstName:"John", lastName:"Doe", age: 46 ,}	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:	**Correcto:**
if (myObj !== null && typeof myObj !== "undefined"	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”.

Convenciones de codificación de JavaScript

Las convenciones de codificación son las directrices de estilo de programación. Las convenciones de codificación se pueden documentar las reglas para los equipos a seguir, o simplemente ser su práctica de codificación individual. Las convenciones mejora la legibilidad del código y hace el mantenimiento del código más fácil.

Nombres de variables: Todos los nombres comienzan con una letra.

Espacios alrededor de Operadores: Siempre ponga espacios alrededor de los operadores (= + / *), y después de las comas.

Sangría del código: Siempre use 4 espacios para la sangría de bloques de código.

Reglas Declaración: Siempre termine declaración simple con un punto y coma.

Para declaraciones más complejas:

- Ponga el soporte de abertura en el extremo de la primera línea.
- Utilice un espacio antes del corchete de apertura.
- Ponga el soporte de cierre en una nueva línea, sin espacios iniciales.
- No finalice declaración complejo con un punto y coma.

Reglas de objetos: Los objetos cortos se pueden definir en una sola línea.

Algunas reglas para definir objetos más grandes son:

- Coloque el soporte de la apertura en la misma línea que el nombre del objeto.
- Utilice dos puntos más un espacio entre cada propiedad y su valor.
- Use comillas valores de cadena, no en torno a valores numéricos.
- No añada una coma después de la última pareja propiedad-valor.
- Coloque el soporte de cierre, en una nueva línea, sin espacios iniciales.
- Siempre termine una definición de objeto con un punto y coma

Longitud de la línea <80: Para facilitar la lectura, evite líneas de más de 80 caracteres. Si una sentencia JavaScript no cabe en una línea, el mejor lugar para romperlo, es después de un operador o una coma.

Convenciones de nomenclatura: Utilice siempre el mismo convenio de denominación para todo el código.

Cargando JavaScript en HTML: Utilice la sintaxis simple para cargar scripts externos.

Acceso a elementos HTML: Una consecuencia de la utilización de "desordenados" estilos HTML, podría dar lugar a errores de JavaScript. Si es posible, utilice la misma convención de nombres (como JavaScript) en HTML.

Extensiones de archivo:

- Archivos HTML deben tener un .html extensión (no .htm).
- Archivos CSS deben tener una .css extensión.
- Archivos JavaScript debe tener un .js extensión.

Utilice nombres de archivo en minúscula: Si utiliza una mezcla de mayúsculas y minúsculas, lo que tienes que ser muy consistente. Si se muda de un caso insensible, a un servidor de mayúsculas y minúsculas, incluso los pequeños errores se romperán su web. Para evitar estos problemas, utilice siempre los nombres de archivo en minúsculas (si es posible).

Rendimiento: Las convenciones de codificación no son utilizadas por las computadoras. Estas convenciones debemos utilizarlas para una mejor legibilidad en código.

JSON

JSON es un formato para almacenar y transportar datos. JSON se utiliza a menudo cuando se envían datos desde un servidor a una página web. JSON significa J ava S cripta O bject N otación. Json es ligero de intercambio de datos de formato, independiente del idioma y fácil de entender.

Ejemplo:

```
{"employees":[
    {"firstName":"John" , "lastName" : "Doe" },
    {"firstName":"Anna" , "lastName" : "Smith" },
    {"firstName":"Peter" , "lastName" : "Jones" }
]}
```

Explicacion:

En este ejemplo creamos los datos de Json los tres datos están contenidos en un array y pos supuesto cada dato es un índice del array.

El formato JSON evalúa a objetos JavaScript

El formato JSON es sintácticamente idéntico al código para la creación de objetos de JavaScript. Debido a esta similitud, un programa de JavaScript puede convertir fácilmente datos JSON en objetos nativos de JavaScript.

Sintaxis

Las llaves contienen objetos y los corchetes arrays, los datos se separan por comas y los datos poseen nombre y valor separados por dos puntos (:).

La conversión de un texto JSON a un objeto JavaScript

Esto se puede realizar almacenando en una variable el texto JSON para luego con la funcion JSON.parse() cambiarlo a un objeto de JavaScript.

Ejemplo:

```
var text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
 '{ "firstName":"Anna" , "lastName":"Smith" },' +
 '{ "firstName":"Peter" , "lastName":"Jones" } ]}';

var obj = JSON.parse(text)
```

Explicación:
En el ejemplo anterior lo que vemos en la variable text es texto de JSON el cual estoy creando directamente en JavaScript, luego en la variable obj estoy pasando el texto JSON a un objeto en JavaScript con la funcion JSON.parse(text).

Performance

Para acelerar tú código de JavaScript debes:

Reducir la actividad en Loops: Cada declaración dentro de un bucle se ejecutará para cada iteración del bucle. Búsqueda de declaraciones o asignaciones que se pueden colocar fuera del bucle.

Reducir DOM Acceso: El acceso al DOM HTML es muy lento, en comparación con otras sentencias de JavaScript. Si usted accede al DOM una sola vez y el dato lo almacena en una variable global será más rápido.

Reducir DOM Tamaño: Mantenga el número de elementos en el DOM HTML pequeño. Esto siempre mejorará carga de la página, y la velocidad de renderizado, sobre todo en los dispositivos más pequeños.

Evite variables innecesarias: No cree nuevas variables si no planea guardar valores en ellas.

Delay JavaScript Loading: Poner sus guiones en la parte inferior del cuerpo de la página, permite que el navegador carga la página por primera vez. Una alternativa es utilizar defer = "true" en la etiqueta de script. El atributo defer específica que el script debe ser ejecutado antes de que la página ha terminado de analizar, pero sólo funciona para los scripts externos.

Evitar el uso with: Evite el uso de la palabra clave with. Tiene un efecto negativo en la velocidad.

JavaScript Timing Eventos

Con JavaScript, es posible ejecutar un código en intervalos de tiempo especificados. Esto se llama eventos de tiempo. Los dos métodos principales que se utilizan son:

setInterval (): Ejecuta una función, una y otra vez, a intervalos de tiempo especificados. El primer parámetro de setInterval () debe ser una función. El segundo parámetro indica la longitud de los intervalos de tiempo entre cada ejecución.

Ejemplo:

```
setInterval(function () {alert("Hello")}, 3000);
```

Explicacion:

La funcion mostrara cada 3 segundos el alert.

El método clearInterval() se utiliza para detener nuevas ejecuciones de la función especificada en el método setInterval().

setTimeout () : Ejecuta una función, una vez, después de esperar un número especificado de milisegundos El primer parámetro de setTimeout () debe ser una función. El segundo parámetro indica el número de milisegundos, a partir de ahora, que quiere ejecutar el primer parámetro.

Ejemplo:

```
<button onclick = "setTimeout(function(){alert('Hello')},3000)">Try
it</button>
```

Explicación:

Al hacer click en el botón después de tres segundos el alert aparecerá.

El método clearTimeout() se utiliza para detener la ejecución de la función especificada en el método setTimeout().

JavaScript Errores

El try declaración le permite probar un bloque de código para los errores.

El catch declaración le permite manejar el error.

El throw declaración le permite crear errores personalizados. . El throw es una excepción.

El finally comunicado le permite ejecutar código, después de tratar de atrapar, sin importar el resultado.

Los errores pueden ser los errores cometidos por el programador, los errores debidos a la entrada equivocada, y otras cosas imprevisibles codificación. Las sentencias de JavaScript try y catch vienen en pares.

Factory constructor pattern

El constructor de la fábrica utiliza una función que crea un objeto por sí mismo sin nueva. La herencia se realiza mediante la creación de un objeto padre primero, y luego modificarlo. Métodos y funciones locales son privadas. El objeto debe ser almacenado en el cierre antes de volver si queremos acceder a sus métodos públicos de los locales.