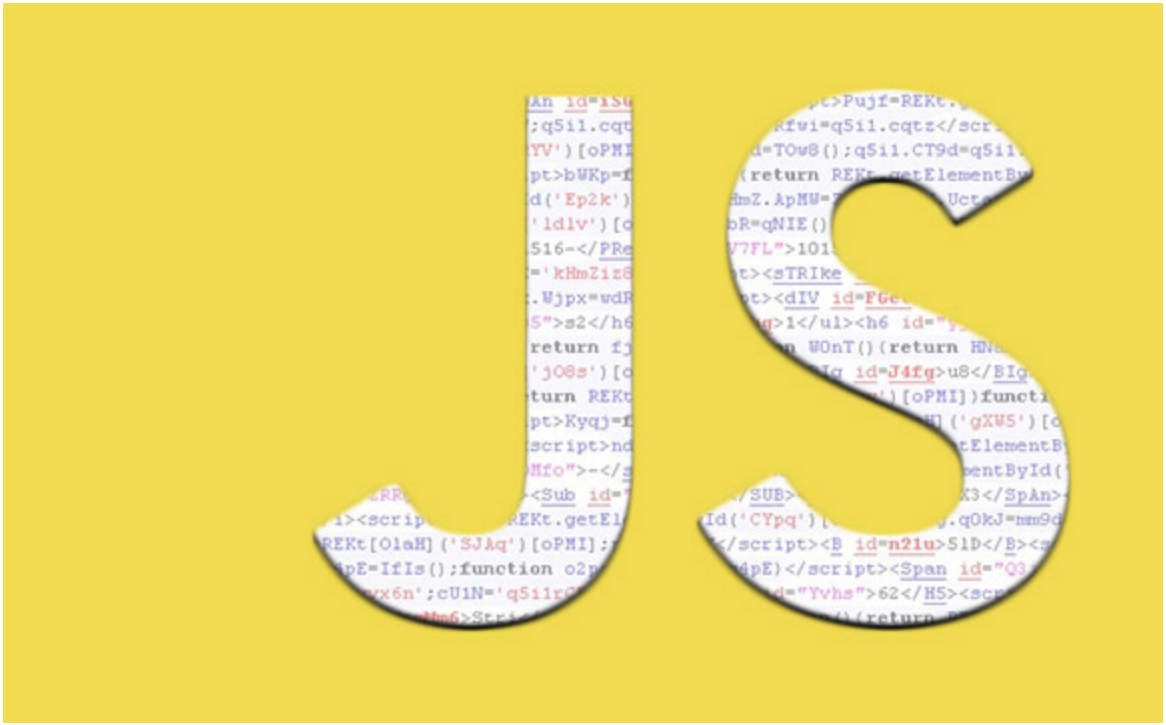
# JavaScript

Kimberly Quirós Jiménez



## 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.

**Prompt:** Muestra un diálogo de campo de formulario con botones Ok y Cancel, un texto definido por el primer parámetro enviado a la función y un input de texto con valor predeterminado definido por el segundo parámetro.

Código:

```
Var name = prompt("Please enter your name:");
```

**Alert:** Crea una caja con un texto que será mostrado al usuario con un botón de aceptar.

Código:

```
Alert("This a custom alert box");
```

**toLowerCase:** Pasa los una cadena de caracteres en minúscula.

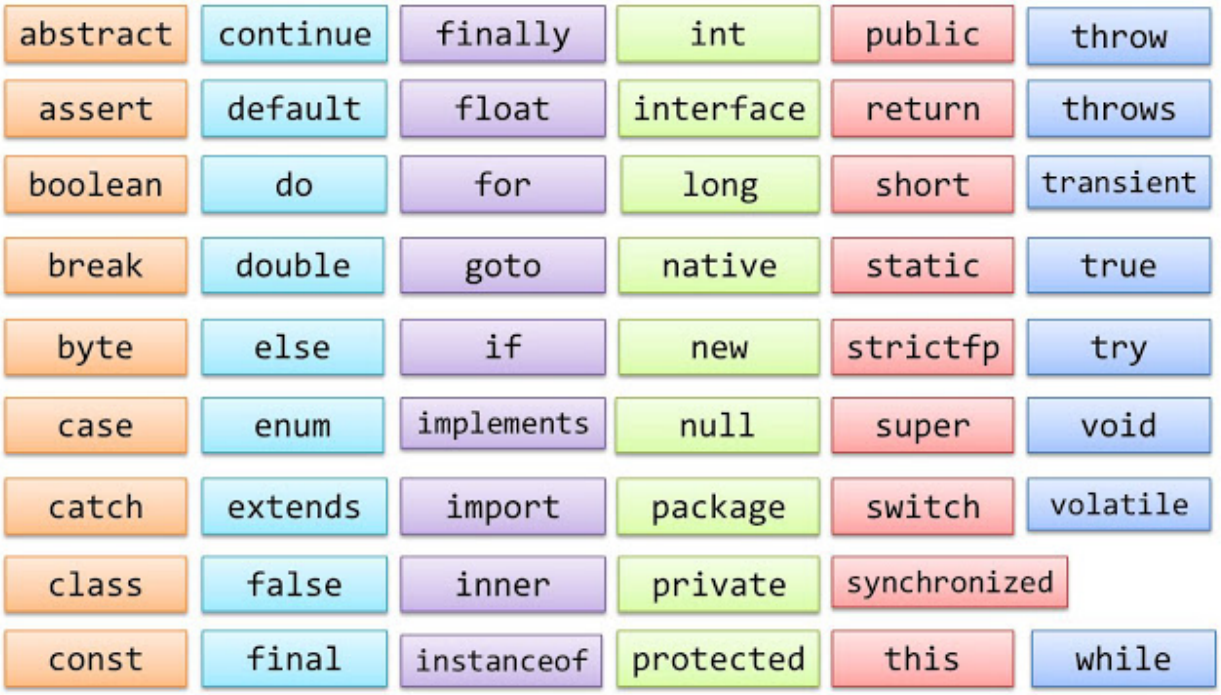
**Eval:** Evalúa una cadena de código.

**Concatenar (números o letras):** Se pueden concatenar cadenas de varias formas en JavaScript.
Lo normal es usar el operador (+).

```
Var name = "Kimberly";
Var lastname = "Quiros";
Var result = name + lastname
```

Se puede concatenar las cadenas de texto que sean necesarias con el operador (+).
También se puede usar la asignación (+=) para concatenar varias cadenas.

## Palabras Reservadas




## Futuras Palabras Reservadas

1. implements
2. interface
3. package
4. private
5. protected
6. public
7. static
8. yield

## Buenas Prácticas:

### Avoid Global Variables:

Minimiza el uso de las Variables Globales. Las variables globales y las funciones pueden ser sobrescritas por otros Script.

### Siempre declarar variables locales:

Todas las variables en una función podrían ser declaradas como variable local.

### Declaraciones arriba:

Esto es una buena práctica al poner todas las declaraciones arriba de cada string o función. Esto hace que sea más fácil evitar variables globales no deseadas.

```
var  num1 , num2;
var  resultado;
num1 =  54;
num2=  3;

resultado = num1 + num2;
```

### Nunca declare String, Number o Booleans como Object:

Siempre trate números, cadenas, o booleanos como valores primitivos. No como objetos.

```
var  x = "John";
var  y = new String("John");
(x === y)
```

## Web browser engine:

> Un **Web browser engine** es software que toma contenido marcado (como HTML, XML, archivos de imágenes, etc.) e información de formateo (como CSS, XSL, etc.) y luego muestra el contenido ya formateado en la pantalla de aplicaciones.

Algunos de los motores de renderizado más notables son:

- Gecko, utilizado en Mozilla Suite, y otros navegadores como Galeon.
- Trident, el motor de Internet Explorer para Windows.
- KHTML/WebCore, el motor de Konqueror. Antecesor del WebKit.
- Presto, el antiguo motor de Opera.
- Tasman, el motor de Internet Explorer para Mac.
- gzilla, el motor de Dillo.
- GtkHTML, el motor de Links.
- WebKit, el motor de Epiphany, Safari.
- Blink, el nuevo motor de Google Chrome y Opera (se trata de un fork de WebKit).
- Servo, nuevo motor en desarrollo por parte de Mozilla (con el apoyo de Samsung), está siendo optimizado para la arquitectura ARM y la plataforma Android.

El término motor de renderizado también puede referirse a motores de renderizado de texto como Pango o Uniscribe los cuales hacen presentables a los textos plurilingües, teniendo en cuenta los textos bidireccionales, combinaciones de "caracteres básicos" con acentos, y otras complicaciones del texto plurilingüe.

V8: Es un motor de código abierto para JavaScript creado por Google. Está escrito en C++ y es usado en Google Chrome. V8 puede funcionar de manera individual (standalone) o incorporada a cualquier aplicación C++.

Client-side: Cualquier lenguaje que se ejecuta en un dispositivo cliente que interactúa con un servicio remoto es un lenguaje del lado del cliente.

Server-side: es el nombre general para los tipos de programas que se ejecutan en el servidor.

Scripting Language: es un lenguaje de programación que soporta las secuencias de comandos, los programas escritos para un entorno especial de tiempo de ejecución que puede interpretar (en lugar de compilar) y automatizar la ejecución de tareas que, alternativamente podría ser ejecutados de uno en uno por un operador humano.

List ECMAScript engine: es el lenguaje de programación estandarizado por Ecma Internacional en la especificación ECMA-262 e ISO / IEC 16262. El lenguaje se usa ampliamente para client-side scripting en la web, en la forma de varias implementaciones conocidas como JavaScript, JScript y ActionScript.

Estos son los motores nuevos de ECMAScript generación de navegadores web, todo implementación justo a tiempo de compilación (JIT) o variaciones de esa idea. Los beneficios de rendimiento para la compilación justo a tiempo hacen que sea mucho más adecuado para aplicaciones web escritas en JavaScript.

Carakan: Un motor de JavaScript desarrollado por Opera Software ASA, incluye en la Comunicado del navegador web Opera 10.50, hasta el cambio a V8 con Opera 15 (lanzado en 2013).
Chakra: Un motor de JScript utilizarse en Internet Explorer. Se fue visto de antemano primero en MIX 10 como parte de Internet Explorer 9 Platform Preview.
SpiderMonkey: Un motor de JavaScript en aplicaciones de Mozilla Gecko, incluyendo Firefox. El motor incluye actualmente los compiladores IonMonkey y JägerMonkey, ha incluido previamente el compilador TraceMonkey (javascript JIT), y está previsto incluir el próximo compilador OdinMonkey.

SquirrelFish: El motor JavaScript de WebKit de Apple Inc.. También conocido como Nitro.
Tamarin: Un motor de ActionScript y ECMAScript usado en Adobe Flash.
V8: Un motor de JavaScript se utiliza en Google Chrome, Node.js y V8.NET.
JavaScriptCore: Un intérprete de JavaScript derivado originalmente de RV. Se utiliza en el proyecto WebKit y aplicaciones como Safari.
Nashorn: Un motor de JavaScript se utiliza en Oracle Java Development Kit (JDK).
Los siguientes motores utilizan intérpretes de tiempo de ejecución, que no se compilan en código máquina nativo y por lo general se ejecutan más lentamente.

Continuum: Un auto-intérprete que soporta las últimas versiones del proyecto de ECMAScript 6 especificación Únicamente, el motor se implementa en ECMAScript 3, lo que hace posible ejecutar ES6 en los navegadores tan antiguos como IE6
Futhark: El motor ECMAScript de las versiones del navegador web Opera 9,50 a 10,10.
Inscript: Una biblioteca propietaria obsoleto utilizado para iCab 2 y 3
JScript: El motor que se utiliza en Internet Explorer para versiones hasta IE9, y uno de los componentes del motor de renderizado Trident.
RV: El motor utilizado en Konqueror, y uno de los componentes de KHTML, un predecesor de JavaScriptCore.
Lineal B: El motor ECMAScript de las versiones del navegador web Opera 7,0-9,50, exclusivo.
Narciso: JavaScript implementado en JavaScript (un evaluador meta-circular), destinado para funcionar en otro motor de JavaScript, de carácter teórico y educativo solamente.
JS-Intérprete un peso ligero intérprete de JavaScript implementado en JavaScript con la ejecución paso a paso.
QtScript: Originalmente desarrollado por Trolltech, ahora propiedad de Digia. Proporciona integración QObject con JavaScriptCore.
Rhino: Uno de los varios motores de JavaScript de Mozilla, utilizando la plataforma Java.
YAJI:. Un motor ECMAScript sobre la base de la FESI aplicación por Jean-Marc Lugrin en 1999, el uso de la plataforma Java, actualmente en desarrollo para apoyar a los últimos estándares (Spec 262, v5.1)
Duktape: una pequeña huella, motor fácilmente integrable ECMAScript E5 / E5.1.
La Plataforma Kinoma, un entorno de ejecución de ECMAScript 5 y el marco

Jsish: un intérprete de Javascript con sqlite orden interna, json, WebSocket y zvfs apoyo.
Websocket.js: motor de Javascript integrable con HTTP / WebSocket apoyo.
Espruino: una muy pequeña huella intérprete específicamente para microcontroladores. Puede funcionar en menos de 8kB de RAM mediante la ejecución de la fuente (en lugar de Bytecode)
MuJS: una biblioteca intérprete ECMAScript ligero, diseñado para encajar en otro software para extenderlos con capacidades de scripting. Originalmente desarrollado para MuPDF.

## Summary:

El operador  precedencia determina el orden en que el operador esta evaluado. El valor con gran altura de precedencia será evaluado de primero.

```
3 + 4 * 5 // returns 23
```

## Windows.onload:

El evento load dispara el evento al final del proceso de carga del documento. En este punto, todos los objetos del documento son DOM,  y todas las imágenes y sub-frames han terminado de cargarse.

```
window.onload = funcRef;
```

FuncRef es la función tipo handler function a ser llamada cuando el evento load de window se dispara.

## Onload:

El atributo onload incendios cuando un objeto se ha cargado.
onload es la más utilizada en el elemento <body> para ejecutar un script una vez a la página web se ha cargado completamente todo el contenido.

El atributo onload se puede utilizar para comprobar el tipo de navegador y versión del navegador del visitante, y cargar la versión correcta de la página web en base a la información.

```
<!DOCTYPE html>
<html>
    <body>
```

```
        <img src="w3html.gif" onload="loadImage()" width="100"
    height="132">

    <script>
        function loadImage() {
            alert("Image is loaded");
        }
    </script>

    </body>
</html>
```

## Onunload:

El onunload atributo incendios una vez al perfil ha descargado (o la ventana del navegador se ha cerrado ).

onunload se produce cuando el usuario se desplaza fuera de la página (haciendo clic en un enlace, envía un formulario, cierre la ventana del navegador , etc. ).

```
<!DOCTYPE html>
<html>
    <body onunload="myFunction()">

        <h1>Welcome to my Home Page</h1>

        <p>Close this window or press F5 to reload the page.</p>
        <p><strong>Note:</strong> Due to different browser settings, this
    event may not always work as expected.</p>

    <script>
        function myFunction() {
            alert("Thank you for visiting W3Schools!");
        }
    </script>
```

```
    </body>
</html>
```

## TypeOf:

El operador typeof devuelve una cadena que indica el tipo del operando sin evaluarlo. Operando es la cadena, variable, palabra clave u objeto para el que se devolverá su tipo. Los paréntesis son opcionales.

El operador typeof devuelve los siguientes resultados operator returns the following results for these variables:

```
typeof miFuncion == 'function'
typeof forma == 'string'
typeof tamano == 'number'
typeof hoy == 'object'
typeof noExiste == 'undefined'
typeof true == 'boolean'
typeof null == 'object'
```

## Null:

El valor nulo es un valor literal JavaScript en representación nula o un "vacío", es decir, sin valor de objeto está presente. Es uno de los valores primitivos de JavaScript.

### Difference between null and undefined

```
 typeof null
typeof undefined
null === undefined
null  == undefined
```

## Undefined:

La propiedad de valor global undefined representa el valor undefined . Es uno de los tipos primitivos de JavaScript.

# Lógica de operadores

La lógica de operadores son típicamente usadas con el valor de boolean.

&& à Cuando se usa con un valor boolean && retorna true si ambos  o perandos son verdaderos; de lo contrario, devuelve false.

|| à Cuando se usa con un valor boolean && retorna true si cualquiera de los operandos son verdaderos; si ambos son falsos, devuelve false.

! à Retorna false si su único operando es verdadero, de lo contrario, devolver false

## Delete:

El operador delete elimina una propiedad de un objeto.

Si el operador delete tiene éxito, se elimina la propiedad del objeto por completo. Sin embargo, si existe una propiedad con el mismo nombre en la cadena de prototipo del objeto, el objeto heredará esa propiedad del prototipo.

## Date:

El objeto Date le permite trabajar con fechas (años, meses, días, horas, minutos,segundos y milisegundos).

JavaScript puede escribirse como una cadena:

Sábado, 18 de abril de 2015 01:11:38 GMT-0600 (Hora estándar, América Central) o como un número:

1429341098764

Las fechas escritas como números, especifica el número de milisegundos desde el 01 de enero de 1970, 00:00:00.

## Contructor:

### Object.prototype.constructor

Retorna una referencia a la función del `Object` que creó el prototipo de la instancia. Note que el valor de esta propiedad es una referencia a la función misma, no a un string conteniendo el nombre de la función. El valor es solo de lectura para valores de primitivas tales como 1, true y 'test'.

```
var o = {};
o.constructor === Object; // true

var a = [];
a.constructor === Array; // true

var n = new Number(3);
n.constructor === Number; // true
```

```
function Type () {}

var types = [
    new Array(),
    [],
    new Boolean(),
    true,              // no cambia
    new Date(),
    new Error(),
    new Function(),
    function () {},
    Math,
    new Number(),
    1,                 // no cambia
    new Object(),
    {},
    new RegExp(),
    /(?:)/,
    new String(),
    "test"             // no cambia
];

for( var i = 0; i < types.length; i++ ) {
    types[i].constructor = Type;
    types[i] = [ types[i].constructor, types[i] instanceof Type, types[i].toS
}

console.log( types.join( "\n" ) );
```

# Funciones parseInt and parseFloat

Las dos funciones "convertidoras", parseInt y parseFloat, retornan un valor numérico cuando se pasa una cadena como un argumento.

## ParseFloat:

Convierte (parsea) un argumento de tipo cadena y devuelve un número de punto flotante.
parseFloat es una función de alto nivel y no está asociada a ningún objeto.
parseFloat convierte su argumento, una cadena, y devuelve un número de punto flotante. Si encuentra un carácter diferente al signo (+ o -), numerales (0-9), un punto decimal o un exponente, devuelve el valor hasta ese punto e ignora ese carácter y todos los correctos siguientes. Se permiten espacios anteriores y posteriores.
Si el primer carácter no se puede convertir a número, parseFloat devuelve NaN.

### Ejemplo:

```
parseFloat("3.14");
parseFloat("314e-2");
parseFloat("0.0314E+2");
var cadena = "3.14"; parseFloat(cadena);
parseFloat("3.14más caracteres no dígitos");
```

## ParseInt:

ParseInt convierte su primer argumento, la cadena cadena, e intenta retornar un entero en una base de raíz especificada, indicada por el segundo parámetro opcional, radix. Por ejemplo, una raíz de 10 indica convertir a número decimal, ocho a octal, dieciseis a hexadecimal y así sucesivamente. Para raíces mayores a diez, las letras del alfabeto indican números mayores a nueve. Por ejemplo, para números hexadecimales (base 16), son utilizadas desde la A hasta la F.
Si parseInt encuentra un caracter que no es un número en la raíz especificada, lo ignora y todos los caracteres sucesivos y retorna el valor entero convertido hasta el punto. Si el primer caracter no puede ser convertido a un número en una raíz especificada, retorna "NaN." La función parseInt trunca la cadena a valores enteros.

### Ejemplo:

## Código:

```
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;
```

## en el browser:

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

## Timing

Es posible ejecutar algún código en intervalos de tiempo especificados. Esto se llama eventos de tiempo .
Es muy fácil para los acontecimientos del tiempo en JavaScript . Los dos métodos principales que se utilizan son :

setInterval () - ejecuta una función , una y otra vez , a intervalos de tiempo especificados
setTimeout () - ejecuta una función , una vez , después de esperar un número

especificado de milisegundos

Nota : El setInterval () y setTimeout () son los dos métodos del objeto HTML DOM ventana.

```
var myVar=setInterval(function () {myTimer()}, 1000);

function myTimer() {
    var d = new Date();
    document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
```

## JS Errors

La sentencia try le permite probar un bloque de código para los errores.

La sentencia catch le permite manejar el error.

La sentencia throw permite crear errores personalizados .

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

Cuando se ejecuta el código JavaScript , pueden ocurrir errores diferentes .

```
function myFunction() {
    var message,     x;
    message = document.getElementById("message");
    message.innerHTML = "";
  x = document.getElementById("demo").value;
     try {
        x = Number(x);
         if(x == "") throw "is empty";
        if(isNaN(x)) throw "is not a number";
        if(x > 10) throw "is too high";
        if(x < 5) throw "is too low";
    }
    catch(err) {
        message.innerHTML = "Error: " + err + ".";
    }
```

```
        finally {
                document.getElementById("demo").value = "";
        }
}
```

## Patrón Constructor Fábrica

Este patrón es especial, ya que no utiliza "new".

The object is created by a simple function call, similar to *Python-style*:

```
var animal = Animal ("fox")
var rabbit = Rabbit ("rab")
```

## Declaración

El constructor se define como una función que devuelve un nuevo objeto:

```
function Animal(name) {
        return {
                run: function() {
                        alert(name + " is running!")
                }
        }
}
```

## herencia

Rabbit se hace mediante la creación de un animal y , a continuación, la mutación es :

```
function Rabbit(name) {
        var rabbit = Animal(name) // make animal

        rabbit.bounce = function() { // mutate
        this.run()
        alert(name + " bounces to the skies! :)")
        }
        return rabbit // return the result
```

```
}
        var rabbit = Rabbit("rab")
        rabbit.bounce()
```

# The Good Parts

## Objects

Los tipos simples son number, string, boolean, null y undefined. Todos los demás valores son objects. Un objeto es un contenedor de propiedades, que contienen un nombre y un valor.

### Object Literals:

Un objeto literal es un par de llaves que rodean cero o más pares nombre / valor.

### Retrieval:

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

### Prototype:

Cada objeto está vinculado a un objeto prototipo de la que puede heredar propiedades. Todos los objetos literales están vinculados a Object.prototype. Un prototipo es utilizado 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.

### 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 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.

## Funciones

Funciones en JavaScript son objetos. Los objetos son colecciones de pares nombre/valor teniendo un enlace oculto a un objeto prototipo. Objetos producidos a partir de los literales de objeto son ligado a Object.prototype. Objetos de función están vinculados a Function.prototype (que es sí mismo ligado a Object.prototype).

Puesto que las funciones son objetos, pueden utilizarse como cualquier otro valor.Funciones pueden ser almacenados en variables, objetos y arreglos de discos. Las funciones se pueden pasar como argumentos para funciones y funciones pueden devolverse de funciones. Además, puesto que las funciones son objetos, las funciones pueden tener métodos.

### Función Literal:

Una función literal tiene cuatro partes.
La primera parte es la función de la palabra reservada.
La segunda parte opcional es el nombre de la función. La función puede utilizar su nombre para
se llama recursivamente.
La tercera parte es el conjunto de parámetros de la función, envuelto en paréntesis.
La cuarta parte es un conjunto de declaraciones envuelto entre llaves.

### Invocación

Invocar una función suspende la ejecución de la función actual, pasando el control y los parámetros de la nueva función. Además de los parámetros declarados, cada función recibe dos parámetros adicionales: this y arguments. El patrón de invocación de método, el patrón de la invocación de la función, la invocación del constructor patrón y el patrón de invocación de aplicar. Los patrones difieren en cómo el bono parámetro que esta está inicializada.

## El patrón de invocación de método

Cuando una función se almacena como una propiedad de un objeto, lo llamamos un método. Cuando un se invoca el método, esto está destinado a ese objeto. Si contiene una expresión de invocación un refinamiento (es decir, a. punto de expresión expresión o [subíndice]), es invocado como un método.

## The Function Invocation Pattern

Cuando se invoca una función con este patrón, esto está enlazado al objeto global.
Esto fue un error en el diseño de la lengua. Había diseñado el lenguaje correctamente, cuando se invoca la función interna, esto estaría todavía limitado aesta variable de la función externa.

## El patrón de invocación de Constructor

JavaScript es un lenguaje de herencia prototipada. Eso significa que los objetos pueden heredar propiedades de otros objetos. La lengua es libre de clase.
Este es un cambio radical de la moda actual. Idiomas la mayoría hoy son clásicos.
Herencia prototipada es poderosamente expresiva, pero no es ampliamente comprendido.
JavaScript sí mismo no está confiado en su naturaleza prototipada, así ofrece un objeto de decisiones sintaxis que recuerda a las lenguas clásicas.

## Argumentos

Es un parámetro de bono que está disponible para las funciones cuando invoca el matriz de argumentos. Da la función acceso a todos los argumentos que fueron proveídos con la invocación, incluyendo exceso argumentos que no fueron asignados a parámetros.

### Retorno

Cuando se invoca una función, se comienza la ejecución con la primera declaración y termina cuando golpea el } que cierra el cuerpo de la función. Que causa la función a la parte del programa que invoca la función de control.

### Excepciones

JavaScript proporciona un mecanismo de control de excepciones. Las excepciones son inusuales (pero
accidentes no totalmente inesperados) que interfieren con el flujo normal de un programa.

### Aumento de los tipos

JavaScript permite que los tipos básicos de la lengua para ser aumentada. En el capítulo 3, nos vi que agregar un método a Object.prototype hace ese método disponible para todos objetos.

### Recursion

A recursive function is a function that calls itself, either directly or indirectly. Recursion is a powerful programming technique in which a problem is divided into a set of similar subproblems, each solved with a trivial solution. Generally, a recursive function calls itself to solve its subproblems.

### Scope

Ámbito de aplicación en un lenguaje de programación controla la visibilidad y la vida de las variables y parámetros.

### Closure

Las buenas noticias acerca de alcance son que las funciones internas acceder a los parámetros y las variables de las funciones están definidas dentro de (con la excepción de esto y argumentos). Esto es algo muy bueno.

# Inheritance

Inheritance es un tema importante en la mayoría de lenguajes de programación. En las lenguas clásicas (como Java), herencia (o amplía) proporciona dos útiles servicios. En

primer lugar, es una forma de reutilización de código. Si una nueva clase en su mayoría es similar a la existente clase, sólo tiene que especificar las diferencias.

### Pseudoclassical

JavaScript está en conflicto sobre su naturaleza prototipada. Su mecanismo de prototipo es oscurecida por complicados negocios sintáctico que parece vagamente clásica.
En lugar de tener los objetos heredan directamente de otros objetos, un nivel innecesario de direccionamiento indirecto se inserta tal que los objetos son producidos por las funciones de constructor.

### Especificadores de objeto

A veces sucede que un constructor se da un gran número de parámetros.
Esto puede ser problemático porque puede ser muy difícil de recordar el orden de los argumentos.

### Protopypal

En un patrón puramente prototipado, prescindir de las clases. Nos centramos en su lugar en el objetos. Herencia prototipada es conceptualmente más simple que la herencia clásica:una nuevo objeto puede heredar las propiedades de un objeto antiguo.

### Funcional

Una debilidad de los patrones de herencia que hemos visto hasta ahora es que no hay privacidad.
Todas las propiedades de un objeto son accesibles. No conseguimos no variables privadas y métodos privados.

## Array

Una array es una distribución lineal de la memoria en el cual los elementos son accedidos por enteros se utilizan para calcular las compensaciones. Las matrices

pueden ser estructuras de datos muy rápido. Por desgracia, JavaScript no tiene nada como este tipo de array

## Array Literals

Literales de conjunto proporcionan una notación muy conveniente para la creación de nuevos valores del array. Un array literal es un par de corchetes de cero o más valores separados por
comas. Array literal puede aparecer en cualquier lugar que puede aparecer una expresión.
.

## Length

Cada array tiene una propiedad length. A diferencia de la mayoría de otros idiomas,array de JavaScript longitud no es un límite superior. Si se guarda un elemento con un subíndice que es mayor igual o inferior a la longitud actual, la longitud aumentará para contener el nuevo elemento. No hay ningún error de límites de array.

## Delete

Dado que los arrays de JavaScript son realmente objetos, puede utilizarse el operador delete para eliminar elementos de un array:
delete numbers [2];
number is ['zero', 'one', undefined, 'shi', ' go'}.
Por desgracia, deja un hueco en el array.

## Enumeración

Dado que los arrays de JavaScript son realmente los objetos, las para en declaración puede utilizarse para iterar sobre todo de las propiedades de una matriz. Por desgracia, para en no hace ninguna garantía acerca de la orden de las propiedades y la mayoría de las aplicaciones de la array esperar los elementos para se produce en orden numérico.

## Métodos

JavaScript proporciona un conjunto de métodos para actuar en arreglos de discos. Los métodos son funciones almacenado en Array.prototype. En el capítulo 3, vimos que Object.prototype puede ser aumentada Array.prototype también puede ser aumentada.

# Métodos

## array.concat(Item...)

El método concat produce una nueva matriz que contiene una copia superficial de esta array con el elementos anexados a ella. Si un elemento es una array, cada uno de sus elementos se anexa individualmente.
También vea array.push(item...) más adelante en este capítulo.

```
var a = ['a', ' b', 'c'];
var b = ['x', 'y', 'z'];
var c = a.concat (b, true);
c es ['a', 'b', 'c', 'x', 'y', 'z', true]
```

## array.Join(separator)

El método de combinación hace una cadena a partir de una array. Esto se logra haciendo una cadena de cada uno de elementos de la array y luego concatenarlos junto con un separador entre ellos.

## array.pop

Los métodos push y pop que una array funcione como una pila. El método pop quita y Devuelve el último elemento de esta array. Si la array está vacía, devuelve undefined.

## array.push

El método push añade elementos al final de una array. A diferencia del método concat, modifica el array y agrega elementos de array entero.

## array.Reverse

El método inverso modifica la matriz invirtiendo el orden de los elementos.

## array.Shift

El método de cambio elimina el primer elemento de una matriz y lo devuelve. Si el array es vacío, devuelve undefined. cambio es generalmente mucho más lento que el pop.

### 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.
.

### array.splice

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 array.

### array.unshift

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)

### 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.

## 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.

### 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.

# Style

Programas de ordenador son las cosas más complejas que hacen que los seres humanos. Los programas son compuesta por un gran número de piezas, expresadas como funciones, las declaraciones y expresiones se organizan en secuencias que deben estar prácticamente libres de error. El tiempo de ejecución comportamiento tiene poco parecido con el programa que lo implementa. El software es generalmente Espera que modificarse en el transcurso de su vida productiva. El proceso de conversión un programa correcto en un programa diferente correcto es extremadamente difícil.

Estas preocupaciones son verdaderas para todos los lenguajes de programación y son especialmente JavaScript. JavaScript flojo escribiendo y tolerancia a errores excesivo proporcionan poco tiempo de compilación aseguramiento de la calidad de nuestros programas, para compensar, nos deberíamos código con estricta disciplina.
JavaScript proporciona apoyo para programas grandes, pero también proporciona formas y expresiones idiomáticas funciona contra grandes programas. Por ejemplo, JavaScript proporciona conveniencias para el uso de variables globales, pero las variables globales se convierten cada vez más problemático como escala programas en complejidad.
Utilizo una sola variable global para contener una aplicación o una biblioteca. Cada objeto tiene su propio espacio de nombres, así que es fácil de usar objetos para organizar mi código.Uso de cierre proporciona más información oculta, aumentando la fuerza de mis módulos.

# Beautiful Features

Simplificado JavaScript no es estrictamente un subconjunto. He añadido algunas novedades. El más simple fue agregar pi como una simple constante. Lo hice para demostrar una característica del analizador.
También he demostrado un mejor reservados palabra política y demostró que reservados palabras son innecesarios. En una función, una palabra no puede usarse como una variable o un parámetro nombre y una característica de la lengua. Se puede utilizar una palabra para uno u otro y el programador puede escoger. Eso me facilita un lenguaje aprender porque no necesitan ser conscientes de las características que no se usa. Y hace más fácil a la lengua ampliar porque no es necesario reservar más palabras para agregar nuevas características