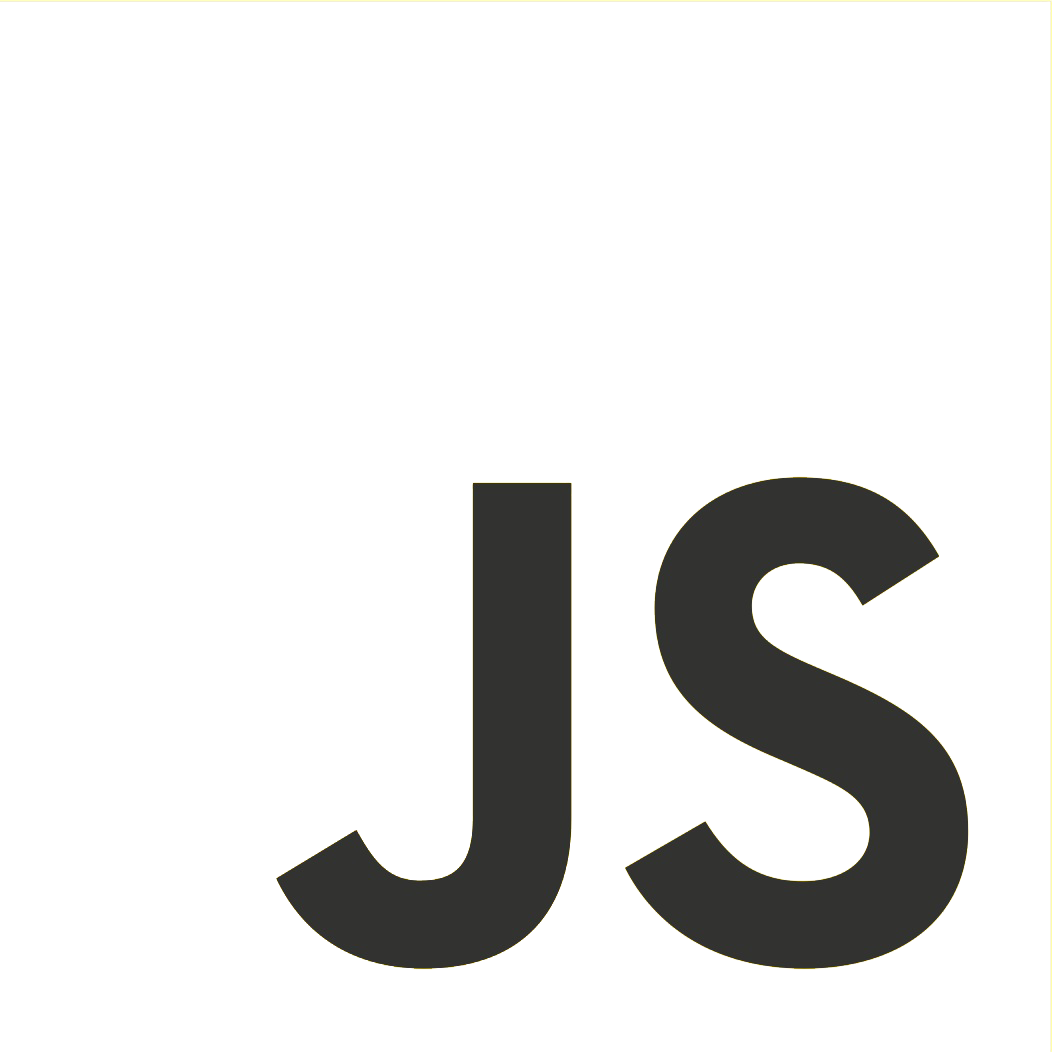
*Lanford Gabriel Murillo*

Resumen final JavaScript intermedio



**¿Qué es JavaScript?**

JavaScript es un lenguaje de scripting multiplataforma, orientado a objetos. JavaScript es un pequeño y liviano lenguaje; no es útil como lenguaje independiente, pero está diseñado para ser fácilmente embebido en otros productos y aplicaciones, como ser web browsers. Dentro de un entorno de desarrollo, JavaScript puede ser conectado a los objetos de este entorno y proveer un control programático sobre ellos.

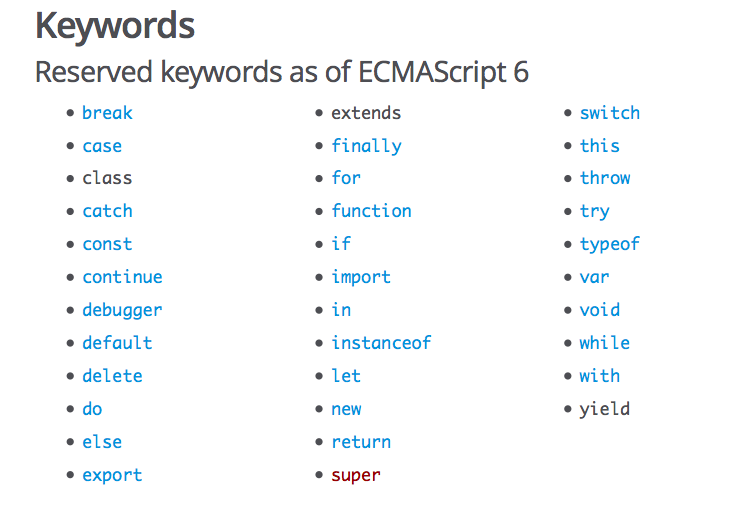
**JavaScript y Java**

Javascript y Java son similares en algunos puntos, pero fundamentalmente diferentes en otros. El lenguaje Javascript se parece al de Java pero no tiene el tipado estático y comprobación de tipos fuerte. Javascript tiene sintaxis, convenciones de nombres y controles básicos de flujo parecidos a los de Java,  por eso es que se le cambio el nombre de Livescript a JavaScript.

En contraste con el sistema de clases construidas por declaraciones en tiempo de compilación de Java, Javascript soporta un sistema en tiempo de ejecución basado en un pequeño número de tipos de datos que representan valores numéricos, booleanos, y cadenas. Javascript tiene un modelo de objetos basado en prototipos en lugar del más común modelo de objetos basado en clases. El modelo basado en prototipos provee herencia dinámica; esto es, lo que se hereda puede variar para objetos individuales. Javascript también soporta funciones sin ningún requerimiento declarativo especial. Las funciones pueden ser propiedades de objetos, ejecutándose como métodos débilmente tipados.



 **Buenas prácticas de JS**



* Evita usar variables globales.
* Siempre declara variables locales.
* Nunca declares: números, strings o booleans como objetos.
* No uses: new Object()
* Usa {} en lugar de new Object().
* Usa “” en lugar de new String.
* Usa 0 en lugar de new Number().
* Usa false en lugar de new Boolean().
* Usa [] en lugar de new Array().
* Usa /()/ en lugar de new RegExp().
* Usa function (){} en lugar de new function().
* Usa el comparador de tipo estricto “===”.
* Evita utilizar eval(). Esta función puede ejecutar texto como código y representa problemas de seguridad.
* Undefined no es lo mismo que null:
* En JS, **null** es para objetos, mientras que **undefined** es para variables.
* Trata de no usar with, ya que tiene un efecto negativo en la velocidad de procesamiento y crea un desorden en el scope de las funciones.

**Funciones en JavaScript**

Las funciones en JS son objetos, y los objetos a su vez, son una colección de pares name/value, los cuales tienen un enlace oculto hacia un objeto prototype*.*

A diferencia de los objetos, que son enlazados a **object.prototype**, las funciones son enlazadas a **function.prototype** (el cual a su vez está enlazado a **object.prototype**). Adicionalmente, cada función es creada con 2 propiedades ocultas más: El contexto de la función, y el código que implementa el comportamiento de la función. Dado que las funciones son objetos, se pueden utilizar como cualquier otro valor.

Las funciones pueden ser almacenadas en variables, objetos y arrays, pueden pasar como argumentos a otras funciones, pueden ser devueltas por funciones. Además, puesto que las funciones son objetos, pueden tener métodos. Y una de las características más sobresalientes de las funciones, es que pueden ser invocadas de diferentes maneras.

Las funciones de 4 partes:

* La palabra reservada **function.**
* El nombre de la función. (El nombre de la función puede servir para: llamarse a sí misma recursivamente; los depuradores y herramientas de desarrollo lo utilizan para identificar a la función. Y si la función no tiene nombre, se dice que es una función ***anónima***).
* Los parámetros de la función. (Envueltos por paréntesis. Dentro de los paréntesis, pueden haber 0 o más parámetros separados por comas, estos nombres serán definidos como variables en la función).
* conjunto de sentencias envueltos entre llaves. Estas declaraciones son el cuerpo de la función y se ejecutan cuando se invoca la misma.

**Invocación de funciones**

Invocar una función suspende la ejecución de la función actual, pasando el control y parámetros a la nueva función. Además de los parámetros declarados, cada función recibe dos parámetros adicionales: **this** y **arguments**. El parámetro **this** es muy importante en la programación orientada a objetos, y su valor se determina por el patrón de invocación. Hay cuatro patrones de invocación en JavaScript:

* El patrón de invocación por **método**.
* El patrón de invocación de **función**.
* El patrón de invocación de **constructor**.
* El patrón de invocación **apply**.

Los patrones difieren la forma en la que el parámetro **this** es inicializado.

El operador de invocación consiste en un par de paréntesis que siguen a cualquier expresión que produce un valor de función. Los paréntesis pueden contener cero o más expresiones, separadas por comas. Cada expresión produce un valor de argumento. Cada uno de las valores de los argumentos serán asignados a los nombres de parámetros de la función. Si hay demasiados argumentos, los adicionales serán ignorados. Si hay muy pocos, el valor **undefined** será sustituido para los valores perdidos. No hay ninguna comprobación de tipos en los argumentos, cualquier tipo de valor puede pasarse a cualquier parámetro.

**Invocación por método**

Cuando una función es almacenada como propiedad de un objeto, lo llamamos: **método**. Cuando un método es invocado, dicho método está ligado al objeto.

Un método puede usar **this** para acceder al objeto recuperar los valores del objeto o modificarlo. La unión de este con el objeto pasa en el **tiempo de invocación**. Esta unión tardía hace a las funciones que utilizan **this** altamente reusables. Métodos que consiguen su contexto del objeto de **this**, son llamados métodos públicos.

**Invocación de función**

Cuando una función no es propiedad de un objeto, es invocada solamente como función.

Cuando la función es invocada de esta forma, **this** es ligado al objeto global. Esto es el resultado de un error en el diseño del lenguaje.