 **U.A.N.L.** 

**Facultad de Ciencias Físico Matemático**

**Materia: Diseño Orientado a Objetos**

**Maestro: Miguel Ángel Salazar Santillán**

**“Semana 2, Cosas raras acerca de JavaScript”**

**Alumno: José Israel Guerrero Ibarra**

**Matrícula: 1734152**

**26/01/2017**

JavaScript se ha vuelto más y más popular en los últimos años, el cual cuenta con una gran interfaz de aplicación, y puedes hacer lo que tu desees, desde programar videojuegos 3D, transmitir video y audio, procesar archivos, entre muchas otras cosas.

Pero, para la gente que apenas está iniciando en el nuevo mundo del lenguaje JS, es difícil comprender en profundidad el lenguaje en sí mismo. Así que en esta ocasión se hablarán de algunas cosas extrañas que suceden en cuanto a la sintaxis, la semántica, entre otras.

**1. “Null” es un objeto**

Null aparentemente es un objeto, el cual, lo que respecta las contradicciones, está arriba, con los mejores. Podemos pensar que Nulo no es un objeto, aunque ciertamente la definición de nulo es la ausencia total de algún valor, podrías estar bien. Pero así es, aquí está la prueba

alert(typeof null); //alerts 'object'

Null no está considerado como instancia de un objeto**,** entonces, si nulo es la ausencia de algún valor, por consiguiente no puede ser una instancia de algo.

**2. “NaN” es un número**

¿Tu pensabas que “Nulo”, que es un objeto, es ridículo? Intenta meterte la idea de que **Nan** es un número. Además, **Nan** no es igual que si mismo.

alert(typeof NaN); //alerts 'Number'

alert(NaN === NaN); //evaluates false

Por último, **NaN** no es igual a nada. La única manera de confirmar que algo es **Nan** es mediante la función isNaN() .

**3. Un arreglo sin llaves es igual a Falso (sobre lo verdadero y lo falso)**

alert(new Array() == false); //evaluates true

En JavaScript, cada valor no-booleano ha sido construido en la bandera booleana, la cual es llamada cuando se pregunta si el valor es booleano.

Debido a que las manzanas no pueden compararse con las peras, cuando se le pide a JavaScript que compare valores de diferentes tipos de datos, primero los "coacciona" a un tipo de datos común. False, zero, null, undefined, cadenas vacías y NaN todos terminan convirtiéndose en falsos - no permanentemente, sólo para la expresión dada.

var someVar = 0;

alert(someVar == false); //evaluates true

**4. Igual no siempre es Igual a**

Miremos esta situación:

1 == 1; //true

'foo' == 'foo'; //true

[1,2,3] == [1,2,3]; //false

Esto es completamente normal, uno es igual a 1, y ‘foo’ es ‘foo. El arreglo [1,2,3] no es igual a [1,2,3] porque simplemente es un tipo de referencia. Eso es común para la mayoría de los lenguajes, pero que pasaría si…

new Array(3) == ",,"; //true

Verdadero nuevamente? Si, es verdadero, y hay una razón lógica para eso… veamos al String representando un nuevo arreglo:

new Array(3).toString(); //",,"

JavaScript implícitamente convierte los valores en ambos lados del operador de igualdad para Strings, y los compara. Este es una de las tantas cosas que deberíamos de tener cuidado. En vez de “==”, es mejor usar “===”. Tres veces igual comprobará incluso el tipo de operandos:

new Array(3) === ",,"; //false

**5. Las funciones pueden ejecutarse ellas mismas**

No podemos negarlo:

(function() { alert('hello'); })(); //alerts 'hello'

La sintaxis es simple: declaramos una función e inmediatamente la llamamos justo en el momento que hacemos lo mismo con otras funciones, con la sintaxis de “( )”. Y quizá te estarás preguntando porque se puede realizar esto. Es como si fuera una contradicción de términos: una función normalmente contiene código que nosotros queremos ejecutar después, no en el momento, de otra manera, nosotros no tendríamos que poner el código en una función.

Un buen uso para la auto ejecución de funciones es enlazar los valores de las variables para usarlos en líneas de código diferentes, como por ejemplos llamadas a eventos, timeouts e intervalos.

**6. 0.1 + 0.2 no es igual a 0.3**

Esto no solo sucede en JavaScript, es actualmente un problema prevaleciente en las ciencias de las computadoras, y esto afecta a muchos lenguajes. El resultado que se obtiene de esta operación es 0.30000000000000004.

Esto es debido a un problema llamado **precisión de máquina**. Cuando JS intenta ejecutar la línea antes mencionada, convierte los valores a sus equivalentes en binarios.

**7. Indefinido puede ser definido**

Aunque pueda sonar raro, *indefinido* no es actualmente una palabra reservada en JavaScript, incluso se piensa que tiene un significado especial y es solo la manera de determinar si una variable es indefinida, así que:

var someVar;

alert(someVar == undefined); //evaluates true

Hasta el momento todo está normal, pero:

undefined = "I'm not undefined!";

var someVar;

alert(someVar == undefined); //evaluates false!

**8. Errores de comparación**

Empecemos por hacer que “x” sea un arreglo con 0 posiciones en él.

js> x=[0] [0]

Debería de ser igual a sí mismo, lo cual está bien.

js> x==x true

Pero a la vez no es igual a si mismo, lo cual empieza a estar mal.

js> x==!x true

Y qué hay acerca de comparar un arreglo de 3 elementos y un string?

js> Array(3)==",," true

**9. Comparar objetos**

Creemos 2 objetos y mantengamos sus referencias en 2 variables:

var o1 = {},

o2 = {};

Tenemos 2 referencias diferentes, así que:

o1 == o2 //false

o1 === o2 //false

Nada raro por el momento. Todos conocemos las matemáticas básicas, así que sabemos que:

a >= b ^ a <= b <=> a = b

Ahora intentémoslo en JavaScript:

o1 >= o2 //true

o1 <= o2 //true

o1 != o2 //true

o1 !== o2 //true

Extraño, no lo es?

**10. replace( ) puede aceptar una llamada de regreso**

Este es uno de los secretos mejores guardados de JavaScript que llegaron en la versión 1.3. La mayoría del uso de “**replace( )**” se mira algo como esto:

alert('10 13 21 48 52'.replace(/d+/g, '\*')); //replace all numbers with \*

Esto es un remplazamiento simple: un string, un asterisco. ¿Pero que si nosotros quisiéramos más control sobre cómo y cuándo nuestros remplazamientos toman lugar? ¿Qué si quisiéramos remplazar solo números debajo del 30? Esto no puede realizarse con expresiones regulares solamente. Necesitamos usar una función de llamada de regreso que evalué cada caso.

alert('10 13 21 48 52'.replace(/d+/g, function(match) {

return parseInt(match) < 30 ? '\*' : match;

}));

Para cada caso echo, JS llama nuestra función, pasando cada número en nuestro argumento. Entonces, nosotros regresamos el asterisco (si el número encontrado es menor a 30) o el número en si.