UNIVERSIDAD AUTONOMA DE NUEVO LEÓN

Facultad de Ciencias Físico Matemáticas

Lic. en Seguridad de las Tecnologías de la Información

Diseño Orientado a Objetos

*“10 cosas extrañas que pasan en JavaScript”*

**Nombre:** Jorge Samuel Ruan Monsivais

**Matricula:** 736099

**Grupo:** 007

**Aula:** 409

**Maestro:** Miguel Salazar

**10 COSAS EXTRAÑAS QUE PASAN EN JAVASCRIPT**

1. Aparentemente null es un objeto siendo en si una contradicción ya que la definición de null sería la total ausencia de valor.

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

Null no es considerado una instancia de objeto. (Los valores en Javascript son instancias del objeto base que las compone. Por lo que cada número es una instancia del objeto Number, cada objeto es una instancia del objeto Object) Esta afirmación nos devuelve a la realidad porque ya que null no posee valor a su vez no puede ser la instancia de nada.

alert(null instanceof Object); //evaluates to false

1. El concepto NaN (not a number) siendo un número. A más a más NaN se considera que no es igual a sí mismo.

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

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

De hecho NaN no es igual a nada. La única manera que tenemos para confirmar que NaN es alguna cosa es a través de la función isNaN().

1. En javascript, cada valor 'no-boleano' tiene una marca boleana integrada que entra en acción cada vez que le pedimos que actúe como un boleano, por ejemplo cuando lo comparamos con un boleano. Cada vez que en javascript tenemos que comparar valores que sean de diferentes tipos de datos primero los forzamos a que tomen un valor de los tipos ya existentes en el lenguaje. False, zero, null, undefinied, strings vacíos y NaN suelen convertirse en False -no permanentemente.
2. La mayoría de usos de  replace() sería algo así:

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

Como podrás ver estamos cambiando todos los strings por un asterisco. Pero y si queremos tenemos un mayor control? Y si queremos cambiar solo los números menores de 30? Esto lo podemos conseguir con tan solo expresiones regulares (tratamos con strings no con mates), pero necesitamos una función para evaluar cada coincidencia.

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

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

}));

Por cada coincidencia que javascript se encuentre llamaremos a la función, pasando cada coincidencia como argumento.

1. El javascript independiente (el que se ejecuta fuera de una función) opera en el ámbito global del objeto window, el cual todo el mundo tiene acceso, en contrapartida las variables locales declaradas dentro de una función solo son accesibles dentro de esa función.

var animal = 'dog';

function get\_animal(adjective){alert(adjective+' '+this.animal);}

get\_animal('lovely'); //alerts 'lovely dog'

La variable y la función están declaradas en el ámbito global (p.e window). Por eso siempre apunta al ámbito actual, en este caso window. Por lo tanto, la función busca window.animal y lo encuentra. Pero en realidad podemos hacer que nuestra función piense que se está ejecutando en un ámbito diferente. Lo hacemos usando el método call() en vez de llamar a la propia función. Esencialmente, call() pretende que nuestra función sea un método de my\_obj.

1. Esta rareza no solo afecta a Javascript sino que es un problema en las ciencias de la computación, afecta a varios lenguajes. El resultado es 0.300000000000004. Ahora entra en juego el concepto de precisión de la máquina. Cuando javascript intenta ejecutar la línea de código convierte los valores a sus equivalentes binarios y aquí es donde empieza el problema. Los valores al ser traducidos en binario pierden su valor original pero son casi idénticos al original.
2. En realidad undefinied no es una palabra reservada en javascript, a pesar de que tiene un significado especial y es la única manera de determinar si una variable es indefinida o no.
3. Muchos desarrolladores de JavaScript intermedios sobreviven sólo en match y replace con las expresiones regulares. Pero JavaScript define más métodos que estos dos. De particular interés es test(), que funciona como matchla excepción de que no vuelva partidos: simplemente confirma si un patrón coincide . En este sentido, es computacionalmente más ligero.
4. JavaScript sólo sabe ámbito de la función. Esto significa que bloquea o bucles for y while no introducen un nuevo ámbito de las variables con eficacia haciendo lo mismo que declarar todas las variables al comienzo de la función. Funciones dentro de otras funciones, pueden acceder a las variables de su ámbito primario (cierre), pero no a la inversa
5. Firefox lee y retorna los colores en RGB en vez de HEX, Nunca he entendido realmente por qué Mozilla hace esto. Sin duda, se da cuenta de que cualquier persona interrogar colores calculadas a través de JavaScript está interesada en formato hexadecimal y no RGB.