Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Diseño Orientado a Objetos

Semana #2 JavaScript

Profesor:

Miguel Angel Salazar

Estudiante:

Angel Adolfo Pacheco Mazuca

1656991

Null es un objeto

El valor null es un literal de Javascript que representa un valor nulo o "vacío". Es uno de los valores primitivos de Javascript.

```
// foo no existe, no está definido y nunca ha sido inicializado:
> foo
"ReferenceError: foo is not defined"

// foo existe, pero no tiene tipo ni valor:
> var foo = null; foo
"null"
```

NaN es un número

Si con null ya nos hemos sorprendido, tratemos de comprender el concepto NaN(not a number) siendo un número. A más a más NaN se considera que no es igual a si 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().

array() '==' False es True

```
Entrada

var arr = [];
console.log('Array:', arr);
if (arr) console.log("It's true!");
if (arr == false) console.log("It's false!");
if (arr && arr == false) console.log("...what??");

Salida

Array: []
It's true!
It's false!
...what??
```

El ==operador hace Tipo de conversión para sus operandos, en este caso los dos lados se convierten en número, las medidas adoptadas en el Resumen Igualdad Comparación Algoritmo serían:

== objeto booleano

== número de objetos

== cadena de número

== número de serie

La función replace() acepta como parámetro funciones callback

El método replace() halla un emparejamiento entre una expresión regular y una cadena, y reemplaza la subcadena emparejada con una nueva subcadena.

```
cadena.replace(regexp|substr, newSubStr|function[,  flags]);
```

(callback) en inglés lo indica es una "llamada de vuelta" y este es un concepto importante al momento de escribir código. Es simple: llamo a una funcion y le envío por parámetro otra función (un callback) esperando que la función que llamé se encargue de ejecutar esa función callback.

Las expresiones regulares se pueden testear con test() además de con match()

expresiones regulares

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto. En JavaScript, las expresiones regulares también son objetos.

test()

```
var cadena = "hello world!";
var result = /^hello/.test(cadena);
console.log(result); // true
```

match()

```
cadena = "Para más información, vea Capítulo 3.4.5.1";
expresion = /(capítulo \d+(\.\d)*)/i;
hallado = cadena.match(expresion);
console.log(hallado);
```

Puedes falsear el alcance de una variable o función

```
var animal = 'dog';
function get_animal(adjective){alert(adjective+' '+this.animal);}
get_animal('lovely'); //alerts 'lovely dog'
```

En el caso de arriba, 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:

```
var animal= 'dog';
function get_animal(adjective){alert(adjective+' '+this.animal);};
varmy_obj = {animal:'camel'};
get_animal.call(my_obj, 'lovely'); //alerts 'lovely camel'
```

Las funciones se pueden ejecutar a si mismas

declaramos una función e inmediatamente la llamamos de la misma manera que llamamos a las otras funciones, con ()

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

Firefox no lee y devuelve los colores en hexadecimal sino en RGB

```
Hello, world!
<script>
var ie = navigator.appVersion.indexOf('MSIE'!= -1);
var p = document.getElementById('some_para');
alert(ie ? p.currentStyle.color : getComputedStyle(p, null).color);</script>
```

la mayoría de navegadores mostraran ff9900 mientras que Firefox devuelve rgb(255,153,0)

0.1 + 0.2 '!==' 0.3

El resultado es 0.300000000000004.

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.

Soluiones posibles:

- Convertirlos a enteros, hacer el cálculo y después convertirlos a decimales.
- Utilizando la lógica para usar un rango de valores en vez de un resultado específico.

En vez de hacer esto:

```
var num_1 = 0.1, num_2 = 0.2, should_equal = 0.3;
alert(num_1 + num_2  should_equal); //false
```

Hacer esto:

```
alert(num_1 + num_2 > should_equal - 0.001 && num_1 + num_2 < should_equal +
0.001); //true
```

Undefined puede ser definido, es decir, que no es una palabra reservada

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 Entonces.

```
undefined = "I'm not undefinied";
var some_var;
alert(some_var == undefined); //evaluates false
```

Bibliografía

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/null

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/NaN

http://stackoverflow.com/questions/4226101/conflicting-boolean-values-of-an-empty-javascript-array

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String/replace

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/RegExp/test

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String/match

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions

http://byverdu.es/lo-que-nadie-explica-acerca-de-javascript-10-rarezas-secretos-y-errores-comunes/