Universidad Autónoma de Nuevo León
Facultad de Ciencias Físico Matemáticas

Unidad de aprendizaje: Diseño Orientado a Objetos

Profesor: Miguel Salazar

Alumno: José Julio Mancha Hernández

Matricula:1727375

Correo Electrónico: josejulio2119@gmail.om

viernes, 27 de enero de 2017

Cosas extrañas de Javascript

1) Warm up

Para entrar en el estado de ánimo hablamos de objetos y que las propiedades del objeto se puede acceder mediante el. (punto) o [] (corchetes) donde el operador punto sólo acepta nombres de variable JavaScript válidos y los corchetes pueden tomar cualquier cadena:

```
var person = {
 name: 'david',
 '&weird property': 'YYCJS'
}

var prop = 'name';

person.name // -> David
person['name'] // -> David
person["name"] // -> David
person[prop] // -> David

person['&weird property'] // -> YYCJS

// ERROR
person.&weird property
```

2) Funcion de Argumentos

En una función argumentos es una variable especial de tipo array que contiene todos los parámetros que se pasan a ella:

```
function sum() {
  var sum = 0;
  for(var i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
sum(1, 2, 3, 4); // -> 10
```

3) Indefinido pude ser definifo

Por extraño que parezca, indefinido no es una palabra reservada en JavaScript, aunque tiene un significado especial y es la única manera de determinar si una variable es indefinida.

Asi que:

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

Hasta ahora, tan normal. Pero

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

4) replace() Puede aceptar una función de devolución de llamada

Este es uno de los secretos mejor guardados de JavaScript y llegó en v1.3. La mayoría de los usos de replace () se parecen a esto:

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

Se trata de un simple reemplazo: una cadena, un asterisco. Pero ¿y si queríamos más control sobre cómo y cuándo suceden nuestros reemplazos? ¿Qué pasa si queremos reemplazar sólo los números menores de 30? Esto no puede lograrse con expresiones regulares solas (son todas sobre cuerdas, después de todo, no matemáticas). Tenemos que saltar a una función de devolución de llamada para evaluar cada partido.

```
alert('10 13 21 48 52'.replace(/d+/g, function(match) {
        return parseInt(match) < 30 ? '*' : match;
}));
```

Para cada partido realizado, JavaScript llama a nuestra función, pasando el partido en nuestro argumento de partido. Luego, devolvemos el asterisco (si el número coincide con menos de 30) o el partido en sí (es decir, no debe tener lugar ningún partido).

5) Scope

JavaScript solo se conoce la función scope. Esto significa que los for o los bucles for y while no introducen un nuevo ámbito variable efectivamente haciéndolo igual que declarando cada variable al principio de la función.
Las funciones dentro de otras funciones pueden acceder a las variables desde su ámbito principal (cierre), pero no al contrario.

```
var x = 'outer';
function count() {
  for(var i = 0; i < 10; i++) {
    var x = 'testing';
    var inner = function() {
      var i = 42;
      // -> x === 'testing'
      // -> i === 42
    }
    inner();
    // -> i === current count
  }
}
// -> x === 'outer'
```

6) NaN es un numero

¿Pensabas que ser un objeto nulo era ridículo? Trate de lidiar con la idea de NaN - "no un número" - ser un número! Por otra parte, NaN no se considera igual a sí mismo

```
alert(typeof NaN); //alerts 'Number'
alert(NaN === NaN); //evaluates false
```

7) Null es un objeto

Al parecer, Null es un objeto, que, en lo que respecta a las contradicciones, es lo mejor de ellos. ¿Nulo? ¿Un objeto? "Ciertamente, la definición de nulo es la ausencia total de valor significativo", usted dice. Tendrías razón. Pero así es. Aquí está la prueba:

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

8) Igualdad

Justo en los talones de la verdad y la falsedad es la comparación de la igualdad. Los operadores básicos == (igual) y! = (No iguales) comparan solamente el valor (sin el tipo), por ejemplo:

```
1 == 1 // -> true
1 == '1' // -> true
1 == 2 // -> false
// Now things get weird
'' == false // -> true
[] == false // -> true, [] not falsy though
null == undefined // -> true
```

Los operadores === (triple igual) y! == (no triples iguales?) Comparan el valor y el tipo:

```
1 === 1 // -> true
1 === '1' // -> false
1 === parseInt('1') // -> true
[] === false // -> false
null === undefined // -> false
```

Los objetos son siempre comparados por su referencia, de modo que la expresión sólo será verdadera si es realmente el mismo objeto:

```
var person = { name: 'David' };
var otherPerson = person;
person === { name: 'David' } // -> false, created new object
person === otherPerson // -> true, same reference
```

9) Que es esto ?

En cualquier función, se trata de una palabra clave reservada que hace referencia al propietario de la función. Por ejemplo, cuando se llama a una función definida en un objeto, será el propio objeto:

```
var person = {
 name: 'David',
 sayHi: function() {
  return 'Hello ' + this.name;
 }
}
```

```
person.sayHi(); // -> 'Hello David'
```

Esto también se utiliza en las funciones del manejador de eventos, donde se refiere al elemento (s) que desencadenó el evento:

```
document.getElementById('mybutton').addEventListener('click',
  function() {
   this.innerHTML = 'Button clicked';
  });
```

10)Cierres

Veamos el siguiente fragmento de código jQuery que crea diez botones, añade un controlador de clics y los agrega al cuerpo.

```
for(var i = 0; i < 10; i++) {
  var button = $('<button>Button #' + i + '</button>');
  button.click(function() {
    alert(i);
  });
  $('body').append(button);
}
```

Haciendo clic en cualquier botón sin embargo no alertará este número de los botones como usted esperaría pero el número 10 para cada botón en lugar de otro.

La razón es de nuevo el alcance variable. La función de control de clics accede a la misma variable de su padre y como siempre se ejecuta después de que se realiza el bucle, tendrá el mismo valor en todas partes.

El truco aquí es introducir un nuevo ámbito a través de una función wrapper y llamarla con el recuento actual. A continuación, se copiará en un nuevo contador de variables:

```
for(var i = 0; i < 10; i++) {
  var button = $('<button>Button #' + i + '</button>');
  var wrapper = function(counter) {
    button.click(function() {
      alert(counter);
    });
   }
  wrapper(i);
  $('body').append(button);
}
```