UT6 – TRABAJANDO CON OBJETOS

6.3.0.- Introducción

JavaScript está diseñado en un paradigma basado en objetos:

- Un objeto es una colección de propiedades, y **una propiedad es** una asociación entre un nombre y un valor.
- Un valor de propiedad puede ser una función, la cual es conocida entonces como **un método** del objeto.

Además de los objetos que están predefinidos en el navegador, se pueden definir objetos propios.

6.3.1.- Visión general sobre los Objetos

Los objetos en JavaScript, al igual que en muchos otros lenguajes de programación, pueden ser comparados con objetos de la vida real. El concepto de Objetos en JavaScript se puede entender como en la vida real, objetos tangibles.

En JavaScript,

un **objeto** es un entidad independiente con propiedades y tipos.

Compárelo con una taza, por ejemplo. Una taza es un objeto, con propiedades. Una taza tiene un color, un diseño, peso, un material del que fue hecho, etc. De la misma manera, los objetos de JavaScript pueden tener propiedades, que definen sus características.

6.3.1.1.- Objetos y propiedades

Un objeto de JavaScript tiene propiedades asociadas. **Una propiedad de un objeto** puede ser explicada como una variable que se adjunta al objeto. Las propiedades de un objeto son básicamente lo mismo que las variables comunes de JavaScript, excepto por el nexo con el objeto. Las propiedades de un objeto definen las características de un objeto. Tú accedes a las propiedades de un objeto con una simple notación de puntos:

```
nombreObjeto.nombrePropiedad
```

Como todas las variables de JavaScript, tanto **el nombre del objeto** (que puede ser una variable normal) **y el nombre de propiedad son sensible a mayúsculas y minúsculas**. **Podemos definir propiedades asignándoles un valor**. Por ejemplo, vamos a crear el objeto **miAuto** y darle propiedades denominadas **marca**, **modelo**, y **annio** así:

```
var miAuto = new Object();
miAuto.marca = "Ford";
```

```
miAuto.modelo = "Mustang";
miAuto.annio = 1969;
```

En JavaScript también **se puede acceder a, o establecer, propiedades de objetos mediante la notación de corchetes [ ].**
Los objetos son llamados a veces ***arreglos asociativos***, ya que cada propiedad está asociada con un valor de cadena que puede ser utilizada para acceder a ella. Así, por ejemplo, se puede acceder a las propiedades del objeto **miAuto** de la siguiente manera:

```
miAuto["marca"] = "Ford";
miAuto["modelo"] = "Mustang";
miAuto["annio"] = 1969;
```

El nombre de la propiedad de un objeto puede ser:

- cualquier cadena válida de JavaScript, o
- cualquier cosa que se pueda convertir en una cadena, incluyendo una cadena vacía.
- cualquier nombre de propiedad que no sea un identificador válido de JavaScript sólo podrá ser accedido utilizando la notación de corchetes (por ejemplo, el nombre de alguna propiedad que tenga un espacio o un guión, o comienza con un número).
- **La notación de corchetes es**, también, **muy útil cuando los nombres de propiedades deben ser determinados de forma dinámica** (cuando el nombre de la propiedad no se determina hasta su tiempo de ejecución). Como en el siguiente ejemplo:

```
var miObjeto = new Object(),
    cadena = "miCadena",
    aleatorio = Math.random(),
    objeto = new Object();
miObjeto.type = "Sintaxis con punto";
miObjeto["Fecha de creación"] = "Cadena con espacios y acento";
miObjeto[cadena] = "String value";
miObjeto[aleatorio] = "Número Aleatorio";
miObjeto[objeto] = "Objeto";
miObjeto[""] = "Incluso una cadena vacía";

console.log(miObjeto);
```

También **se puede acceder a las propiedades mediante el uso de un valor de cadena que se almacena en una variable**:

```
var nombrePropiedad = "marca";
miAuto[nombrePropiedad] = "Ford";

nombrePropiedad = "modelo";
miAuto[nombrePropiedad] = "Mustang";
```

Se puede utilizar la notación de corchetes con for ... in para repetir las propiedades de un objeto **enumerable**. Para ilustrar cómo funciona esto, la siguiente función muestra las propiedades del objeto cuando se pasan como argumentos de la función **el objeto** y **el nombre del objeto**:

```
function mostrarPropiedades(objeto, nombreObjeto) {
 var resultado = "";
 for (var i in objeto) {
  if (objeto.hasOwnProperty(i)) {
     resultado += nombreObjeto + "." + i + " = " + objeto[i] + "\n";
  }
 }
 return resultado;
}
```

Por lo tanto, la llamada a la función **mostrarPropiedades (miAuto, "miAuto")** retornaría lo siguiente:

```
console.log(mostrarPropiedades(miAuto, "miAuto") );
miAuto.marca = Ford
miAuto.modelo = Mustang
miAuto.annio = 1969
```

6.3.2.- Todo como un objeto

En JavaScript, casi todo es un objeto.

Todos los tipos primitivos, excepto **null** y **undefined,** se tratan como objetos. Pueden asignar propiedades (propiedades asignadas de algunos tipos no son persistentes), y tienen todas las características de los objetos.

6.3.2.1.- Listando todas las propiedades de un objeto

A partir de ECMAScript 5, hay tres formas nativas de "Lista/cruzada" de propiedades de objeto:

- bucles for...in
  Este método atraviesa todas las propiedades enumerables de un objeto y su cadena de prototipo
- Object.keys(o)
  Este método devuelve una matriz con los mismos nombres ("keys") enumerables de las propiedades del objeto "o" (no en la cadena de prototipos).
- Object.getOwnPropertyNames(o)
  Este método devuelve una matriz que contiene todos los nombres (enumerables o no) de las propiedades del objeto "o".

Antes de ECMAScript 5, **no había** una **forma nativa de listar todas las propiedades de un objeto**. Sin embargo, esto se puede lograr con la siguiente función:

```
function listaTodasLasPropiedades(o){
  var objetoAInspeccionar;
  var resultado = [];

  for(objetoAInspeccionar = o; objetoAInspeccionar !== null;
objetoAInspeccionar =
Object.getPrototypeOf(objetoAInspeccionar)){
    resultado =
resultado.concat(Object.getOwnPropertyNames(objetoAInspecciona
r)) + "\n";
  }
  return resultado;
}
```

Esto puede ser útil para revelar las propiedades "ocultas" (propiedades de la cadena de prototipo que no se puede acceder a través del objeto, porque otra propiedad tiene el mismo nombre antes en la cadena de prototipo). La lista de propiedades accesibles sólo se puede hacer fácilmente mediante la eliminación de duplicados en la matriz.

6.3.2.3.- Creando nuevos objetos

JavaScript tiene un número de objetos predefinidos.
Además, JavaScript permite crear tus propios objetos. En JavaScript 1.2 y versiones posteriores, puedes **crear un objeto:**

- **usando un inicializador de objeto**,
- **puedes crear primero una función constructora** y luego crear una instancia de un objeto con esa función y el operador new.
- **Usando el método Object.create**

El uso de inicializadores de objeto

El uso de los inicializadores de objeto se refiere a veces a cómo crear objetos con la notación literal. "Inicializador de objeto" es consistente con la terminología utilizada por C + +.

La **sintaxis para un objeto usando un inicializador** de objeto es:

```
var objeto = { propiedad_1:   valor_1,   // propiedad_# puede ser un identificador...
        2:            valor_2,   // o un numero...
        // ...,
        "propiedad n": valor_n }; // o una cadena
```

donde **objeto** es el nombre del nuevo objeto,
cada **propiedad_i** es un identificador (ya sea un nombre, un número o una cadena literal), y
cada **valor_i** es una expresión cuyo valor se asigna a la **propiedad_i**.

El objeto y la asignación es opcional; si usted no necesita hacer referencia a este objeto desde otro lugar, no necesita asignarlo a una variable. (Tenga en cuenta que tal vez necesite envolver el objeto literal entre paréntesis si el objeto aparece donde se espera una declaración, a fin de no confundir el literal con una declaración de bloque.)

Si un objeto se crea con un inicializador de objeto en un script de nivel superior:

- JavaScript interpreta el objeto cada vez que se evalúa una expresión que contiene el objeto literal, y
- Además, se crea un inicializador de función cada vez que se llama a la función.

La siguiente declaración crea un objeto y lo asigna a la variable **x** si y sólo si la expresión cond es **true**.

```
if (cond) var x = {hola: "allí"};
```

El siguiente ejemplo crea miHonda con tres propiedades. Observa que la propiedad del motor es también un objeto con sus propias propiedades.

```
var miHonda = {color: "rojo", ruedas: 4, motor: {cilindros: 4,
tamanio: 2.2}};
```

También puedes utilizar inicializadores de objetos para crear matrices. Consulta array literals. Ya visto en tema anterior.

Crear objetos usando Funciones Constructoras

En JavaScript 1.1 y versiones anteriores, no se puede utilizar inicializadores de objeto.

Puedes crear objetos usando sólo sus funciones constructoras o utilizando una función suministrada por algún otro objeto para ese propósito. Consulta Using a constructor function.

Pasos para crear un objeto usando una función constructora:

1. Definiendo el tipo de objeto al escribir una función constructora. Hay una convención fuerte, con buena razón, para utilizar una letra inicial **capital** (mayúscula).
2. Cree una instancia del objeto con el operador **new**.

Para **definir un tipo de objeto**, puedes crear una función del tipo de objeto que especifica su nombre, propiedades y métodos.

Por ejemplo:

Supongamos que se desea crear un tipo de objeto para los coches. Se quiere que este tipo de objeto se llame auto, y deseas que tenga propiedades de marca, modelo y el año. Para ello, se podría escribir la siguiente función:

```
function Auto(marca, modelo, annio) {
        this.marca = marca;
        this.modelo = modelo;
        this.annio = annio;
}
```

Observa el uso de la cláusula "**this**" para asignar valores a las propiedades del objeto en función de los valores pasados a la función.

Ahora puedes crear un objeto llamado miAuto de la siguiente manera:

```
var miAuto = new Auto("Eagle", "Talon TSi", 1993);
```

Esta declaración crea miAuto y le asigna los valores especificados para sus propiedades. Entonces el valor de:

miAuto.marca es la cadena "Eagle",
miAuto.annio es el número entero 1993,

y así respectivamente.

Puedes crear cualquier número de objetos Auto con las llamadas a **new**. Por ejemplo,

```
var kenscar = new Auto("Nissan", "300ZX", 1992);
var vpgscar = new Auto("Mazda", "Miata", 1990);
```

Un objeto puede tener una propiedad que es, en sí mismo, otro objeto. Por ejemplo, supón que defines un objeto llamado persona de la siguiente manera:

```
function Persona(nombre, edad, sexo) {
        this.nombre = nombre;
        this.edad = edad;
        this.sexo = sexo;
}
```

y luego creas una instancia de dos nuevos objetos persona de la siguiente manera:

```
var fer = new Persona("Fernando Duclouk", 38, "M");
var alvaro = new Persona("Alvaro Caram", 36, "M");
```

Entonces, puedes volver a escribir la definición de automóvil para incluir una propiedad **propietario** que tomará el objeto **persona**, de la siguiente manera:

```
function Auto(marca, modelo, annio, propietario) {
        this.marca = marca;
        this.modelo = modelo;
        this.annio = annio;
```

```
        this.propietario = propietario;
}
```

Para crear instancias de los nuevos objetos, a continuación, utiliza lo siguiente:

```
var auto1 = new Auto("Eagle", "Talon TSi", 1993, fer);
var auto2 = new Auto("Nissan", "300ZX", 1992, alvaro);
```

Note que en lugar de pasar un valor de cadena o entero literal cuando se crean los nuevos objetos, las declaraciones anteriores pasan los objetos **fer** y **alvaro** como argumentos para propietario. Si luego quieres saber el nombre del propietario del **auto2**, puedes acceder a la siguiente propiedad:

```
auto2.propietario.nombre
```

Ten en cuenta que **siempre se puede añadir una propiedad a un objeto previamente definido**. Por ejemplo, la declaración

```
auto1.color = "negro";
```

agrega un color de propiedad del auto1, y le asigna el valor "negro". Sin embargo, esto no afecta a ningún otro objeto.
Para agregar la nueva propiedad a todos los objetos del mismo tipo, hay que añadir la propiedad a la definición del tipo de objeto de auto.

Usando el método Object.create
Este **método** puede ser muy útil, ya que le **permite elegir el objeto prototipo del objeto que desea crear**, sin tener que definir una función constructora.
Para obtener información más detallada sobre el método y la forma de usarlo, consulte Object.create method

Herencia
Todos los objetos en JavaScript heredan al menos otro objeto. El objeto que **se heredó de** es conocido como el **prototipo**, y las propiedades heredadas se pueden encontrar en el objeto prototipo del constructor.

6.3.3.- Propiedades del objeto indexado

En **JavaScript 1.0**, puede hacer referencia a una propiedad de un objeto:

- por su nombre de la propiedad o
- por su índice ordinal.

Sin embargo en **JavaScript 1.1** y posteriores:

- si inicialmente definimos una propiedad por su nombre, debe referirse siempre a ella por su nombre, y
- si inicialmente definimos una propiedad por un índice, siempre debe referirse a ella por su índice.

Esta restricción es de aplicación cuando se crea un objeto y sus propiedades con una función constructora (como hicimos antes con el tipo de objeto de Auto) y al definir propiedades individuales de forma explícita (por ejemplo, miAuto.color = "rojo").
Si inicialmente define una propiedad de objeto con un índice, como miAuto[5] = "25 mpg", puede hacer referencia posteriormente a la propiedad sólo como miAuto[5].

La **excepción a esta regla son los objetos reflejados de HTML**, como por ejemplo la matriz de formularios. Siempre se puede hacer referencia a objetos en estas matrices,

- ya sea por su número ordinal (con base en donde aparecen en el documento) o
- por su nombre (si está definida).

Por ejemplo, si la segunda etiqueta <FORM> en un documento tiene un atributo de nombre "myForm", puede referirse al formulario como document.forms [1] o document.forms ["myForm"] o document.myForm.

6.3.4.- Definición de las propiedades de un tipo de objeto

Usted puede *agregar una propiedad a un tipo de objeto definido previamente* mediante el uso de la propiedad **prototype**. Esto define una propiedad que es compartida por todos los objetos del tipo especificado, en lugar de por una sola instancia del objeto.

El código siguiente agrega una propiedad de color a todos los objetos del tipo de auto, y luego asigna un valor a la propiedad color del objeto auto1.

```
Auto.prototype.color = null;
auto1.color = "negro";
```

Para más información, ver la propiedad prototype del objeto Function en la Referencia de JavaScript.

6.3.5.- Definiendo los métodos

Un **método es una función asociada a un objeto**, o, simplemente, un **método es una propiedad de un objeto que es una función**. Los métodos se definen normalmente como una función, con excepción de que tienen que ser asignados como la propiedad de un objeto.
Ejemplos son:

```
nombreDelObjeto.nombreDelMetodo = nombre_de_la_funcion;

var miObjeto = {
  miMetodo: function(parametros) {
    // ...hacer algo
  }
};
```

donde **nombreDelObjeto** es un objeto existente,
nombreDelMetodo es el nombre que se le va a asignar al método,
y **nombre_de_la_funcion** es el nombre de la función.

Entonces puede llamar al método en el contexto del objeto de la siguiente manera:

```
object.nombreDelMetodo(parametros);
```

Se pueden definir métodos para un tipo de objeto, incluyendo una definición del método, en la función constructora del objeto.
Por ejemplo, se podría definir una función que formatée y muestre las propiedades de los objetos de automóviles previamente definidos; por ejemplo,

```
function mostrarAutos() {
  var resultado = "Un bonito " + this.marca + " " + this.modelo
    + " " + this.annio;
  imprimir_con_estilo(resultado);
}
```

donde **imprimir_con_estilo** es una función para mostrar una línea horizontal y una cadena. Observe el uso de ésta para referirse al objeto al que pertenece el método.
Usted puede hacer de esta función un método de auto mediante la adición de la declaración

```
this.mostrarAutos = mostrarAutos;
```

a la definición del objeto.

Por lo tanto, la definición completa de auto ahora se vería así

```
function Auto(marca, modelo, annio, propietario) {
 this.marca = marca;
 this.modelo = modelo;
 this.annio = annio;
 this.propietario = propietario;
 this.mostrarAutos = mostrarAutos;
}
```

Entonces puedes llamar al método **mostrarAutos** para cada uno de los objetos de la siguiente manera:

```
auto1.mostrarAuto();
auto2.mostrarAuto();
```

Esto produce el resultado que se muestra en la siguiente figura.

```
Un bonito Eagle Talon TSi 1993
Un Bonito Nissan 300ZX 1992
```

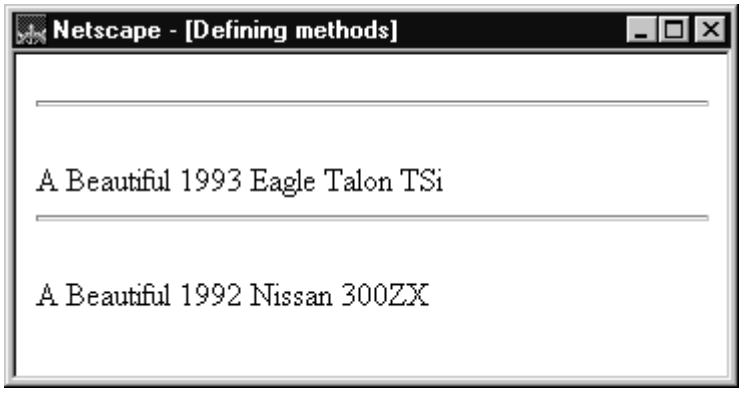

Figura 7.1: Muestra el resultado del método.

6.3.6.- Usando this para las referencias a objetos

JavaScript tiene una palabra clave especial, **this**, que **se puede utilizar dentro de un método para referirse al objeto actual**.
Por ejemplo, supongamos que tenemos una función llamada “validar” que valida el valor de la propiedad de un objeto, teniendo en cuenta al objeto y los valores altos y bajos:

```
function validar(objeto, valoralto, valorbajo) {
 if ((objeto.value < valoralto) || (objeto.value > valorbajo))
   alert("Valores no válidos!");
}
```

Entonces, se podría llamar a **validar** en el controlador de eventos **onchange** de cada elemento del formulario, usando **this** para pasarle el elemento, como en el siguiente ejemplo:

```
<input type="text" name="edad" size="3"
 onChange="validar(this, 18, 99)">
```

En general, **this** **se refiere al objeto de llamada en un método**.
Cuando lo combinamos con la propiedad del formulario, **this** puede referirse al objeto actual del formulario principal.
En el siguiente ejemplo, el formulario **miForm** contiene un objeto de texto y un botón. Cuando el usuario hace clic en el botón, el valor del objeto de texto se establece en el nombre del formulario. El manejador de eventos del botón onclick utiliza this.form para referirse al formulario principal, myForm.

```
<form name="miForm">
<p><label>Nombre del formulario:<input type="text" name="text1"
value="Beluga"></label>
<p><input name="button1" type="button" value="Mostrar Nombre
del Formulario"  onclick="this.form.text1.value = this.form.name">
</p>
</form>
```

Definiendo getters y setters

- Un **getter** es un método que obtiene el valor de una propiedad específica.
- Un **setter** es un método que establece el valor de una propiedad específica.

Puede definir **getters** y **setters** de cualquier objeto predefinido del núcleo o de un objeto definido por el usuario que admita la adición de nuevas características. La sintaxis para definir getter y setters utiliza la sintaxis literal de un objeto.

JavaScript 1.8.1 Nota:
A partir de JavaScript 1.8.1, los setters ya no son llamados a la hora de establecer las propiedades en los objetos y matrices inicializadores.

Estos son reconocidos únicamente en FireFox.

6.3.7.- Eliminando propiedades

Puedes **eliminar una propiedad no heredada** mediante el operador **delete**. El siguiente código muestra cómo eliminar una propiedad.

```
//Crea un nuevo objeto, miobjeto, con dos propiedades, a y b.
var miobjeto = new Object;
miobjeto.a = 5;
miobjeto.b = 12;

//Elimina la propiedad, dejando miobjeto con sólo la propiedad b.
delete miobjeto.a;
console.log ("a" in miobjeto) // yields "false"
```

También puedes utilizar **delete** para eliminar una variable global si la palabra clave **var** no fue utilizada para declarar la variable:

```
g = 17;
delete g;
```

Vea delete para más información.

6.3.8.- Comparando Objetos

Como sabemos los objetos son tipo de referencia en JavaScript.
Cuando comparamos dos objetos que hacen referencia al mismo objeto devolverá true.
Comparando dos objetos que se ven exactamente igual, me refiero a ambos objetos que tienen los mismos métodos y propiedades, devolverá false.

```
// variable de referencia del objeto fruta
var fruta = {nombre: "manzana"};

// variable de referencia del objeto fructificar
var fructificar = {nombre: "manzana"};

fruta == fructificar // retorna false
fruta === fructificar // retorna false
```

Nota: El operador "===" se utiliza para comprobar el valor así como el tipo, ejemplo:
1 === "1" // retorna false
1 == "1"   // retorna true

```
// variable de referencia del objeto fruta
var fruta = {nombre: "manzana"};
```

```
// variable de referencia del objeto fructificar
var fructificar = fruta; // asignamos la referencia del objeto fruta a la
//variable de referencia del objeto fructificar

// aquí fruta y fructificar apuntan al mismo objeto llamado fruta
fruta == fructificar // retorna true

// aquí fruta y fructificar apuntan al mismo objeto llamado fruta
fruta === fructificar // retorna true
```