CFGS DAW

Mòdul 6: Desenvolupament web en entorn client

INS Joan d'Àustria

Objetos

El concepto de objeto va asociado al de clase. Los lenguajes orientados a objetos se definen tradicionalmente como lenguajes que permiten crear múltiples objetos a través de una clase.

ECMA-262 define un objeto como una "colección desordenada de propiedades cada una de las cuales contiene un valor primitivo, un objeto o una función"

Objetos

La forma más simple de crear un objeto es creando una nueva instancia del elemento Object y añadirle propiedades y métodos. También se puede usar la notación literal.

```
var person = new Object();
```

```
var person = {};
```

Propiedades

Las propiedades hacen referencia a la información que queremos almacenar de los objetos.

```
var homer= new Object();
homer.nombre = “Homer Simpson”;
homer.edad = 34;
homer.empleo = “Ingeniero nucelar”;
```

```
var homer= {
        nombre:  “Homer  Simpson”,
        edad : 34,
        empleo:  “Ingeniero nucelar”
};
```

Utilizando la notación literal, crea un nuevo objeto llamado yo con las mismas propiedades que el objeto person y asigna los valores que quieras en estas propiedades.

Propiedades

Para acceder a una propiedad de un objeto podemos hacerlo mediante la notación objeto.propiedad o objeto[“propiedad”]

Muestra por pantalla la información del objeto yo que has creado antes utilizando las dos formas de acceso

Métodos

De la misma manera que podemos ver las propiedades como variables referidas a un objeto concreto, podemos ver los métodos como funciones aplicadas a un objeto concreto

```
var homer= new Object();
homer.nombre = “Homer Simpson”;
homer.edad = 34;
homer.empleo = “Ingeniero nucelar”;
homer.setEmpleo = function(nuevo){
        homer.empleo=nuevo;
};
homer.setEmpleo(“Respons. seguridad)”;
```

this

También disponemos de this, que podemos ver como un puntero que apuntará al objeto que llame al método, es decir, el propietario de la llamada.

```
var homer= new Object();
homer.nombre = “Homer Simpson”;
homer.edad = 34;
homer.empleo = “Ingeniero nucelar”;
homer.setEmpleo = function(nuevo){
        this.empleo=nuevo;
};
homer.setEmpleo(“Respons. seguridad)”;
```

Crea un método que permita actualizar la edad. A continuación, llama a ese método de forma que incremente la edad de homer a 35 años.

this

De la misma forma, podemos usar métodos con this que retornen una propiedad del objeto

```
var homer= new Object();
homer.nombre = “Homer Simpson”;
homer.edad = 34;
homer.empleo = “Ingeniero nucelar”;
var homer.getEmpleo= function(){
        return this.empleo;
};
```

Crea un método que permita obtener el año de nacimiento de homer. Este método debe calcular el año de nacimiento a partir de la propiedad edad.

Clases y objetos

En realidad javascript no soporta el concepto de clase, al menos no de la forma que lo hacen otros lenguajes como java o C++. Es posible, sin embargo definir pseudoclases mediante el uso de funciones constructoras y prototipos.

Clases y objetos

Ya hemos visto como crear objetos:

Mediante el literal { }

Con new Object()

El operador new debe ir seguido de una llamada a una función. Así, se crea un objeto vacio y se llama a la función a la que se le pasa este objeto vacio como el valor para el keyword this.

Una función diseñada para ser usada con new se llama constructor. Su objetivo es iniciar un objeto recién creado con una serie de propiedades necesarias antes de que se utilice el objeto.

Clases y objetos

Podemos crear nuestros propios constructores simplemente escribiendo una función que añade propiedades al elemento this. Posteriormente podremos crear objetos llamando a esta función

```
function Rectangle(w, h) {
        this.width = w;
        this.height = h;
}
```

Observar que esta función no retorna nada!

```
var rect1 = new Rectangle(2, 4);
var rect2 = new Rectangle(8.5, 11);
```

rect1 y rect2 son ahora objetos con las propiedades definidas en la función Rectangle

Crea un constructor para crear objetos de la clase persona. Después crea a homer, marge, bart y lisa a partir de este constructor

Clases y objetos

A partir de este constructor, ¿cómo podríamos redefinir los métodos para que sirvan para todos los objetos?

```
function Persona(nom, edad){
	this.nombre=nom;
	this.edad=edad;
	this.getFecha=function(){
		var x=new Date();
		return (x.getFullYear()-this.edad);
	};
}
var homer=new Persona("Homer", 34);
var bart=new Persona("Bart", 8);
alert (homer.getFecha());
alert (bart.getFecha());
```

Objetos y funciones

Los objetos de la clase persona que hemos definido tienen una propiedad edad. ¿Podemos crear una **función** que reciba como parámetros 2 personas y retorne la diferencia de edad entre ellas?

```
function edadDiff(persona1, persona2){
        …
}
```

Objetos y arrays

Los objetos son un tipo más en javascript, como los números, las fechas… Por tanto, podemos crear un array de objetos

```
function Persona(nom, edad){
        this.nombre=nom;
        this.edad=edad;
}
var familia=new Array();
familia[0]=new Persona("Homer", 34);
familia[1]=new Persona("Bart", 8);
var x;
for (x=0; x<familia.length;x++){
   document.write(familia[x].nombre);
}
```

Clases y objetos

Cada objeto javascript incluye internamente una referencia a otro objeto llamado prototype. Cualquiera de las propiedades del objeto prototype simulan ser propiedades del objeto, o de otra manera: un objeto javascript hereda las propiedades de su prototype.

Cuando se crea un objeto con el operador new, no sólo se crea un objeto vacio, sino que también se crea su prototype.

Clases y objetos

Cualquier propiedad que añadamos al objeto prototype aparecerán como propiedades del objeto inicializado por el constructor.

```
function Rectangle(w, h) {
        this.width = w;
        this.height = h;
}
Rectangle.prototype.area = function( ) {
        return this.width * this.height;
};

var uno=new Rectangle(3,5);
alert(uno.area());
```

Clases y objetos

Los objetos prototype no sólo están para los objetos definidos por nosotros. Las clases ya construidas como string o date también los tienen. Así podemos crear nuevos métodos para todos los elementos string

```
String.prototype.endsWith = function(c) {
        return (c == this.charAt(this.length-1))
}
```

```
var message = “Hola Mundo";
message.endsWith(‘a') ;      // false
message.endsWith(‘o') ;      // true
```

Objetos y arrays asociativos

Un array asociativo es el que nos permite usar un string en lugar de un número para acceder a una determinada posición.

```
var normalArray = [];
normalArray[1] = 'Un valor';
alert(normalArray[1]);            //  Un valor

var associativeArray = [];
associativeArray['nombre'] = 'Homer Simpson';
alert(associativeArray['nombre']); // Homer  Simpson
```

Javascript no soporta los arrays asociativos, pero los arrays javascript en realidad son objetos, y los objetos nos permitirán emularlos

Objetos y arrays asociativos

Hemos visto que para acceder a una propiedad de un objeto, utilizamos el ".", aunque también es posible utilizar "[ ]"

```
objeto.propiedad
```

```
objeto["propiedad"]
```

Por esta razón el ejemplo de la transparencia anterior funciona correctamente.

Objetos y arrays

El recorrido por las propiedades de un objeto, se puede realizar entonces de la misma manera que el recorrido por los elementos de un array asociativo:

```
var associativeArray = [];
associativeArray["one"] = “Uno";
associativeArray["two"] = “Dos";
associativeArray["three"] = “Tres";
for (i in associativeArray) {
        document.writeln(i+':'+associativeArray[i]+', ');
                // one:Uno, two:Dos, three:Tres,
};
```

```
for(var propiedad in persona) {
        document.write(propiedad);
}
```

Aspectos avanzados

Hay dos tipos de propiedades: propiedades de datos y propiedades de acceso

Å Las propiedades de datos presentan 4 atributos para modelar su comportamiento:

- ï Configurable: Indica si vamos a poder modificar la propiedad (eliminarla, cambiar el valor del atributo, etc. Por defecto es false.
- ï Enumerable: Indica si la propiedad será retornada en un bucle for..in que recorra las propiedades del objeto. Por defecto es true.
- ï Writable: Indica si se podrá modificar el valor del atributo. Por defecto es true.
- ï Value: Contiene el valor de la propiedad. El valor por defecto es undefined.

Aspectos avanzados

Para modificar el valor de alguna de estas propiedades debemos usar el método Object.defineProperty(objeto, propiedad, valores)

```
var person = {};
Object.defineProperty(person, “name”, {
        writable: false,
        value: “Homer Simpson”
});
alert(person.name); //”Homer Simpson”
person.name = “Bart”;
alert(person.name); //”Homer Simpson”
```

Aspectos avanzados

- Las propiedades de acceso contienen una combinación de función getter y setter.
  - Cuando se accede a alguna propiedad para consultar su valor, no se muestra este, sino que se llama a la función getter en su lugar y esta será la responsable de retornar un valor válido
  - Cuando se accede a alguna propiedad para modificar su valor, se llama a la función setter que será la responsable de decidir cómo tratar la información
  - Nuevamente utilizaremos el método Object.defineProperty para indicar el comportamiento

Aspectos avanzados


```
var book = {
_year: 2004,
edition: 1
};
Object.defineProperty(book, “year”, {
        get: function(){
                return this._year;
        },
        set: function(newValue){
                if (newValue > 2004) {
                        this._year = newValue;
                        this.edition = newValue - 2004;
                }
        }
});
book.year = 2005;
alert(book.edition); //2
```