Objetos literales

El diseño del lenguaje JavaScript se basa en un paradigma de objetos. Estos objetos son elementos dotados de una colección limitada de **propiedades** (características del objeto) y **métodos** (capacidades del objeto, acciones que puede realizar).

Estos son clasificados en dos grupos:

- Los definidos de forma estándar en los navegadores (**window**, **document** o **date**)

- Los personalizados: objetos que podemos diseñar libremente en base a unas necesidades particulares de desarrollo, determinando qué nombre, propiedades y métodos tendrá.

Dentro de estos últimos, el primer tipo de objeto (y el más sencillo) son los objetos de notación literal, más conocidos como *objetos literales*. Esta denominación se debe a que son de baja reusabilidad ya que los valores de cada propiedad son indicados de forma, valga la redundancia, literal.

Sintaxis de un objeto literal

La estructura de un objeto literal está limitada por llaves, donde se encapsula cada identificador asignándole un valor literal, en un formato *clave : valor*.

Los identificadores pueden ser **nombres**, **valores numéricos o valores de cadena de caracteres**, y los valores que les asociemos pueden ser datos de cualquier tipo.

Es según el valor que le asociemos a cada identificador, que hablaríamos de una **propiedad** (cuando asociamos cualquier valor excepto una función) o de un **método** (cuando asociamos una función como valor).

Su sintaxis es la que sigue:

```
var nombreObjeto = {
    identificador1: valor1,
    identificador2: valor2,
    identificador_n: valor_n
}
```

Propiedades en un objeto literal

Las propiedades son **identificadores** que llevan asociado un valor diferente a una función, siguiendo el modelo sintáctico anterior:

```
var factura = {
    numero: 201,
    cliente: 'Transportes Chemita',
    divisa: 'eur',
    subtotal: 350.25,
    IVA: 75.55
}
```

Durante el desarrollo debemos respetar las tabulaciones y saltos de línea que se muestran en estos ejemplos por motivos de mantenibilidad y legibilidad, aunque en su estado minificado los objetos tienen este aspecto:

```
var factura = {numero:201, cliente:'Transportes Chemita', subtotal:350.25, IVA:75.55}
```

El objeto podrá ser entonces accedido con normalidad siguiendo la notación del punto, tanto a efectos de obtención de sus valores como de asignación de nuevos valores sustitutivos:

```
var numeroFactura = factura.numero;
var monedaFactura = factura.divisa;
var subtotalFactura = factura.subtotal;
console.log('La factura ' + numeroFactura + ' es de ' + subtotalFactura + ' ' + moneda

// La factura 201 es de 350.25 eur

factura.numero = 202;   // Sustituimos el antiguo valor de su propiedad "numero"
numeroFactura = factura.numero;
console.log('La factura ' + numeroFactura + ' es de ' + subtotalFactura + ' ' + moneda

// La factura 202 es de 350.25 eur
```

Métodos en un objeto literal

Los métodos de un objeto no son más que identificadores cuyo valor es **una función interna o externa**, dotando al objeto de capacidades específicas (las instrucciones encapsuladas en esa función): las acciones que el objeto puede desempeñar.

Los métodos tienen la capacidad de acceder a los valores de cualquier propiedad que forme parte del mismo objeto, mediante de la palabra clave *this*:

```
var factura = {
    numero: 201,
    cliente: 'Transportes Chemita',
    divisa: 'eur',
    subtotal: 350.25,
    IVA: 75.55,
    total: function(){
      return this.subtotal + this.IVA;
    }
}

var numeroFactura = factura.numero;
var totalFactura = factura.total();
// los métodos, por su capacidad de recibir parámetros, requieren paréntensis en su e

console.log('La factura ' + numeroFactura + ' tiene un importe de ' + totalFactura);

// La factura 201 tiene un importe de 425.8 eur
```

De igual forma, la naturaleza de función de estos métodos les capacita para aceptar parámetros, lo que le da **cierto grado de reusabilidad**. En este caso vemos cómo podemos enviar un descuento a la factura, que se restaría al valor total calculado (siendo cero si no invocamos el método con este parámetro):

```
var factura = {
    numero: 201,
    cliente: 'Transportes Chemita',
    divisa: 'eur',
    subtotal: 350.25,
    IVA: 75.55,
    total: function(incremento){
      var incremento = incremento || 0;
      return this.subtotal + this.IVA - incremento;
    }
}

var totalFactura = factura.total();
var totalFacturaDescuento = factura.total(10);

console.log('La factura asciende a ' + totalFactura + ', y podemos aplicar un descuen

// La factura asciende a 425.8, y podemos aplicar un descuento obteniendo 415.8
```

Los métodos pueden también enlazar funciones externas a su objeto:

```
var factura = {
    numero: 201,
    cliente: 'Transportes Chemita',
    divisa: 'eur',
    subtotal: 350.25,
    IVA: 75.55,
    total: calculaTotal
}

function calculaTotal (incremento) {
    var incremento = incremento || 0;  // Asignamos valor cero por defecto
    return this.subtotal + this.IVA + incremento;
}

console.log( factura.total(10) );

// 435.8
```

Anidación de objetos literales

Es posible contener objetos dentro de otros objetos, creando una rica y compleja estructura de información.

Por ejemplo, podemos almacenar mediante un Array las tres fechas de vencimiento de una factura:

```
var factura = {
    numero: 650,
    cliente: 'Transportes Madrid SL',
    vencimientos: [new Date(2017,5,20), new Date(2017,7,25), new Date(2017,9,21)]
    }
}
```

O podemos, también, crear su equivalencia encapsulándolas en un objeto interno dentro del propio objeto de factura:

```
var factura = {
    numero: 650,
    cliente: 'Transportes Madrid SL',
    vencimientos: {
      1: new Date(2017,5,20),
      2: new Date(2017,7,25),
      3: new Date(2017,9,21)
    }
}

var numeroFactura = factura.numero;
var primerVencimiento = factura.vencimientos[1];
console.log('El segundo vencimiento de la factura ' + numeroFactura + ' será el ' + p

// El segundo vencimiento de la factura 650
```

En los casos que hagamos uso de valores numéricos en vez de nombres para denominar una propiedad, debemos acceder a las mismas como si de un índice se tratara, es decir, mediante corchetes, aunque abusar de esta práctica puede repercutir en la legibilidad e identificabilidad del código.

También podemos usar un String como valor para un identificador, caso en el que también necesitaremos corchetes para accederlo.

No existe un límite en el nivel de profundidad de estas anidaciones. Podríamos almacenar sub objetos con la fecha y el importe de cada vencimiento:

```
var factura = {
    numero: 650,
    cliente: 'Transportes Madrid SL',
    vencimientos: {
      1: {
            fecha: new Date(2017,5,20),
            importe: 216
      },
      2: {
            fecha: new Date(2017,7,25),
            importe: 216
      },
      3: {
            fecha: new Date(2017,9,21),
            importe: 218
      }
    }
}

var numeroFactura = factura.numero;
var importeTercerVencimiento = factura.vencimientos[3].importe;
console.log('El importe del tercer vencimiento asciende a ' + importeTercerVencimiento

// El importe del tercer vencimiento asciende a 218
```

Extensión de nuevas propiedades

La permisividad de Javascript da lugar a asociar nuevas propiedades (o métodos, si le son asignadas valores de función) que no estaban contempladas durante la incialización del objeto, como por ejemplo el estado de pago de la factura, o dotarla de capacidad de impresión:

```
factura.pagada = false;
factura.imprimir = function(){
      document.write('La factura ' + this.numero + ' corresponde al cliente ' + this.c
    }
```

Entonces, el acceso a los valores de otras propiedades mediante this sigue vigente, incluso si pertenece a un método externo, como es el caso anterior:

```
factura.imprimir();

// Escribirá en el documento "La factura 650 corresponde al cliente Transportes Madrid
```