4. Objetos definidos por el usuario

- JavaScript proporciona una serie objetos predefinidos, sin embargo es posible crear nuevos objetos definidos por el usuario.
- Cada uno de estos objetos puede tener sus propios métodos y propiedades.
- Todos los objetos de JavaScript pertenecen a la 'clase' `Object`.
- JavaScript es un lenguaje basado en objetos: hay objetos pero no clases (Class).
- Los objetos predefinidos de JavaScript (Date, Number, String, Array, Function, …) derivan de la 'clase' Object.

4. Objetos definidos por el usuario

- La forma más sencilla de crear un objeto es mediante la palabra reservada new seguida del nombre de la clase que se quiere instanciar:

```
var elObjeto = new Object();
var laCadena = new String();
```

- El objeto `laCadena` creado mediante el objeto nativo String permite almacenar una cadena de texto y la variable `elObjeto` almacena un objeto genérico de JavaScript, al que se pueden añadir propiedades y métodos propios para definir su comportamiento.

4. Objetos definidos por el usuario

- Definición de un objeto:
  - Técnicamente, un objeto de JavaScript es un array asociativo formado por las propiedades y métodos.
  - La forma más directa para definir las propiedades y métodos de un objeto es mediante la notación de puntos de los arrays asociativos.

```
//NOTACION PUNTOS
var elArray = new Array();

elArray.primero = 1;
elArray.segundo = 2;
```

```
//NOTACION TRADICIONAL
var elArray = new Array();

elArray['primero'] = 1;
elArray['segundo'] = 2;
```

4. Objetos definidos por el usuario

- Declaración e inicialización de los objetos:
  - Un objeto es una entidad que posee unas propiedades que lo caracterizan y unos métodos que actúan sobre estas propiedades.
  - En JavaScript la base para crear objetos es el “constructor”: se trata de una función que da nombre a la “clase” (simulación de la clase) y permite inicializar el objeto. Por lo que, no existe la necesidad de definir explícitamente un método constructor.

4. Objetos definidos por el usuario

Constructor: es una función especial para crear un objeto. Empieza por letra mayúscula:

```
function Coche()

{ //Propiedades y métodos }
```

Creación de un objeto:

```
var unCoche = New Coche();
```

4. Objetos definidos por el usuario

- Sintaxis:

```
function mi_objeto (valor_1, valor_2, valor_x){
  this.propiedad_1 = valor_1;
  this.propiedad_2 = valor_2;
  this.propiedad_x = valor_x;
}
```

- Las **propiedades** de nuestro objeto que se creen dentro del constructor, se definirán empleando la palabra reservada `this`, que se utiliza para hacer referencia al objeto actual.

4. Objetos definidos por el usuario

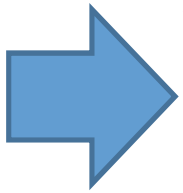
- Ejemplo: Creación del objeto Coche.

"Constructor" sin parámetros:

```
function Coche()
  {this.marca = "" //Vacío
   this.modelo = "" //Vacío
   this.combustible = "Diesel"
//Inicializado
   this.cantidad = 0//Inicializado
}
// Cantidad indica la cantidad de
// combustible inicial.
```

"Constructor" con parámetros:

```
function Coche(marca,modelo,
combustible,cantidad)
  {this.marca = marca;
   this.modelo = modelo;
   this.combustible = combustible;
   this.cantidad = cantidad;
  }
//Cada propiedad toma los valores
// recibidos como parámetros.
```

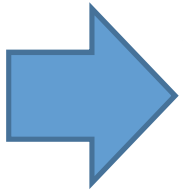


Crear un objeto vacío (sin propiedades):

```
var cocheVacio = New Coche();
```

Cambiar valores en las propiedades:

```
cocheVacio.marca = "Seat";
cocheVacio.maodelo = "Ibiza";
cocheVacio.combustible =
"Diesel";
cocheVacio.cantidad = 40;
```



Crear un objeto inicializado(con propiedades):

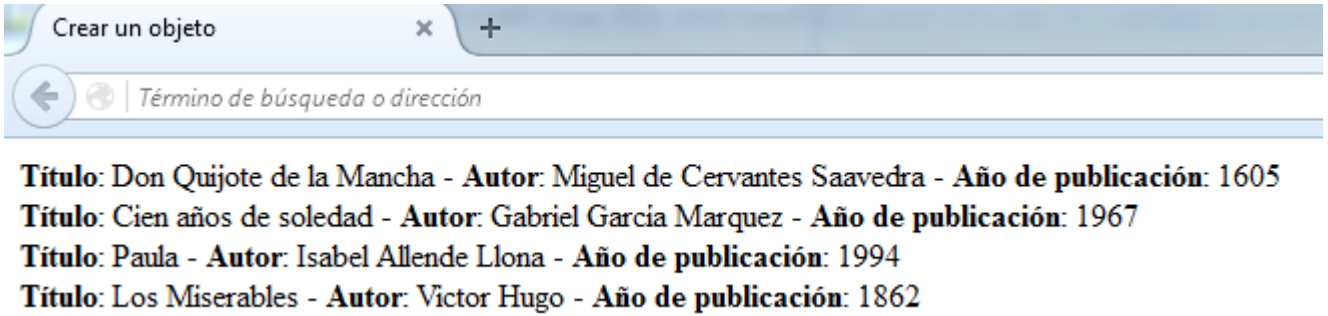
```
var miCoche = New
Coche("Kia","Ceed","Diesel",1);
```

Acceder a las propiedades:

```
Document.write("Mi coche es un:
" + miCoche.marca +
miCiche.modelo);
```

Actividad 5

- Declara un nuevo tipo de objeto llamado Libro que posea solamente tres propiedades (*título, autor y año de publicación*).
- Crea cinco instancias del mismo y almacena cada una de ellas en un array.
- Posteriormente, recorre el array con un bucle para mostrar los datos de cada uno de los libros por pantalla.



4. Objetos definidos por el usuario

- Definir métodos del objeto:
  - Permiten acceder y modificar las propiedades de los objetos. Empiezan por minúscula..

Método que modifica la cantidad de combustible:

```
function rellenardeposito(litros)
  { this.cantidad = litros
  }
```

Referencia al método fuera del objeto (¡no recomendado!):

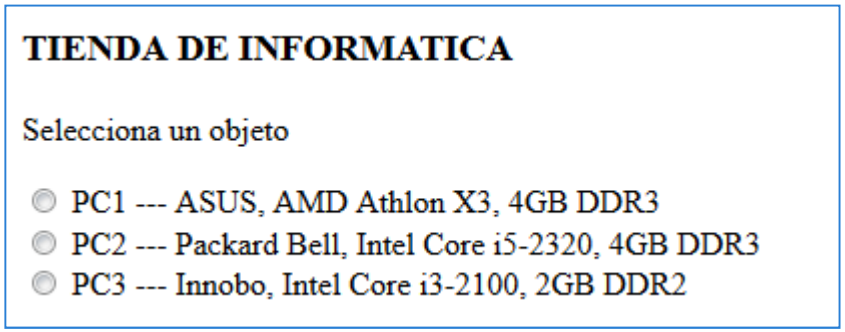
```
function Coche(marca,modelo,
combustible,cantidad)
  { this.marca = marca;
    this.modelo = modelo;
    this.combustible = combustible;

   //Métodos

    this.rellenarDeposito =
          rellenardeposito;
  }
```

Referencia al método dentro del objeto:

```
function Coche(marca,modelo,
combustible,cantidad)
  { this.marca = marca;
    this.modelo = modelo;
    this.combustible = combustible;

   //Métodos

    this.rellenarDeposito = function
             (litros);
       {
          this.cantidad = litros;
       }
  }
```

Actividad 6

- Crea un objeto llamado PC con tres propiedades (marca, cpu y memoria).
- Haz una página donde mostremos tres ordenadores de características diferentes y podamos elegir uno de ellos mediante un radio button.
- Una vez hecha la elección crearemos una instancia del objeto seleccionado y mostraremos sus características mediante un mensaje.



4. Objetos definidos por el usuario

- Los objetos también pueden definirse con el literal, el cual es un valor fijo, formado por parejas de tipo: nombre : valor

```
var coche= {marca:"Kia", modelo:"Ceed", combustible: "Diesel",
cantidad: 40};
```

Ejemplo equivalente:

```
var coche = new Object();
coche.marca = "Kia";
coche.modelo = "Ceed";
coche.combustible = "Diesel";
coche.cantidad = 40;
```

Acceso:

```
coche.marca;
coche[marca];
```

- Aplicar el operador punto sobre undefined o null provoca un error de ejecución y aborta la ejecución del programa.

4. Objetos definidos por el usuario

- La notación array permite acceder también a propiedades cuyo nombre está en una variable en forma de string. (Esto no es posible con la notación punto).

```
var x = {titulo:`Avatar´, director:`James
Cameron´};


x.titulo;       //”Avatar”
x[‘titulo’];    //”Avatar”

var p = ‘titulo’; //inicializada con string ‘titulo’
x[p];           //”Avatar”
x.p;           //undefined
//el objeto x no tiene ninguna propiedad de nombre p
```

4. Objetos definidos por el usuario

- Un objeto independientemente del tipo que sea, posee las propiedades `length`, `constructor`, `prototype` ... y un conjunto de métodos entre los que se encuentran `call()` y `apply()`.
- Ambos métodos permiten ejecutar una función como si fuera un método de otro objeto.

```
function miFuncion(x){
        return this.numero + x;
}

var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.call(elObjeto, 4);
alert(resultado); //9
```

4. Objetos definidos por el usuario

- Método **`call()`:**

```
function miFuncion(x){
        return this.numero + x;
}

var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.call(elObjeto, 4);
alert(resultado); //9
```

- El primer parámetro del método `call()` es el objeto sobre el que se va a ejecutar la función.
- La función externa está implementada como si fuera un método del objeto, ya que la palabra reservada `this` hace referencia al objeto indicado en la llamada a `call()`.
- El resto de parámetros del método `call()` son los parámetros que se pasan a la función, en este ejemplo sólo un parámetro.

4. Objetos definidos por el usuario

- Método **`apply()`**:
- Es idéntico al método `call`(), salvo que en este caso los parámetros se pasan como un array.

```
function miFuncion(x){
        return this.numero + x;
}

var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.apply(elObjeto, [4]);
alert(resultado); //9
```

4. Objetos definidos por el usuario

- **HERENCIA:**
- Un objeto hereda métodos de su clase.
- Por ejemplo: los objetos de la clase Date heredan métodos como toString(), getDay(), getHours(), …

```
var fecha = new Date();

fecha.toString() //Fri Aug 08 2014 12:3436 GTM+0200(CEST)
fecha.getHours() //12
```

- Solo se puede invocar métodos heredados o definidos en un objeto. Invocar un método no heredado ni definido en un objeto provoca error de ejecución.

4. Objetos definidos por el usuario

- **Herencia**:
- La herencia en JavaScript se basa en la utilización de la propiedad `prototype`.
- Por ejemplo: Dada la "pseudoclase" Factura,

```
function Factura(IdFactura, IdCliente){
        this.idFactura = IdFactura;
        this.idCliente = IdCliente;

        this.muestraCliente = function(){
        alert(this.idCliente);}

        this.muestraId = function(){
        alert(this.idFactura);}
}
var laFactura1 = new Factura (1, 8);
laFactura1.muestraCliente();

var laFactura2 = new Factura (2, 5);
laFactura2.muestraId();
```

4. Objetos definidos por el usuario

- En el ejemplo anterior, las funciones `muestraCliente()` y `muestraId()` se crean de nuevo por cada objeto creado. Es decir, cada vez que se instancia un objeto, se definen tantas nuevas funciones como métodos incluya la función constructora.
- Si un método se declara con `this.NombreMetodo` en el constructor, este método no se hereda, es necesario definirlo con el atributo `prototype` para que se pueda heredar:

```
FuncionConstructora.prototype.NombreMetodo
```

(FuncionConstructora equivale a la pseudoclase)

4. Objetos definidos por el usuario

- Ejemplo: ObjetosHerencia.html

```
<script type="text/javascript">
function Persona(){
  //define la pseudoclase Persona
      //metodo camina
      Persona.prototype.camina = function(){
         alert("Persona caminando");}
      //metodo saluda
      Persona.prototype.saluda = function(){
         alert("Persona saludando");}
}
function Alumno(){
  //define la pseudoclase Alumno
}
//se define la herencia
Alumno.prototype = new Persona(); //Alumno hereda de Persona
//reemplazamos el método saluda
Alumno.prototype.saluda = function(){ alert("Hola, soy alumno");}
//Añadimos el método seDespide
Alumno.prototype.seDespide = function(){alert("Alumno dice adiós");}
//Creamos instancia del objeto Alumno
var alumno1 = new Alumno();
alumno1.saluda(); //metodo reemplazado
alumno1.camina(); //metodo heredado de Persona
alumno1.seDespide(); //metodo agregado a la pseudoclase Alumno
//comprueba la herencia
alert(alumno1 instanceof Persona); //true
alert(alumno1 instanceof Alumno); //true
</script>
```

Notación JSON

- JSON (JavaScript Object Notation) es un formato ligero para el intercambio de información. El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.
- La especificación completa de JSON se puede consultar en RFC 4627 (http://tools.ietf.org/html/rfc4627) y su tipo MIME oficial es **application/json**
- Como ya se sabe, la notación tradicional de los arrays es tediosa cuando existen muchos elementos:

```
var modulos = new Array();
modulos[0] = "Lector RSS";
modulos[1] = "Gestor email";
modulos[2] = "Agenda";
modulos[3] = "Buscador";
modulos[4] = "Enlaces";
```

Notación JSON

- El ejemplo anterior se puede reescribir de la siguiente manera utilizando JSON:

```
var modulos = ["Lector RSS", "Gestor email",
"Agenda", "Buscador", "Enlaces"];
```

- Para los **arrays asociativos** se puede utilizar la notación de puntos, pero la notación JSON permite definirlos de una forma mucho más concisa.

```
//NOTACION TRADICIONAL
var modulos = new Array();
modulos.titulos = new Array();
modulos.titulos.['rss'] = "Lector RSS";
modulos.titulos.['email'] = "Gestor email";
modulos.titulos.['agenda'] = "Agenda";

//NOTACION JSON
var modulos = {titulos:{rss:"Lector RSS",
email:"Gestor email", agenda:"Agenda"}}
```

Notación JSON

- Notación JSON genérica para crear arrays y objetos:
- Arrays: (corchetes [])

```
var array = [valor1, valor2, valor3,..., valorN];
```

- Objetos (arrays asociativos): (llaves {})

```
var objeto = {clave1:valor1, clave2:valor2,
clave3:valor3,..., claveN:valorN};
```

- Manera habitual de definir objetos en JavaScript mediante JSON:

```
var objeto = {
  propiedad1: valor_simple_1,
  propiedad2: [array_valor1, array_valor2],  //array
  propiedad3: {propiedad_anidada: valor}, //objeto
  metodo1: nombre_funcion_externa,
  metodo2: function(){ ... }, //función anónima
};
```

Notación JSON

- Ejemplo: Objeto Libro
- Notación tradicional:

```
var Libro = new Object();
  Libro.numeroPaginas = 150;
  Libro.autores = new Array();
  Libro.autores[0] = new Object();
  Libro.autores[0].id = 50;
  Libro.autores[1] = new Object();
  Libro.autores[1].id = 67;
```

- Notación JSON:

```
var Libro = {numeroPaginas:150, autores:[{id:50},
{id:67}]};
```

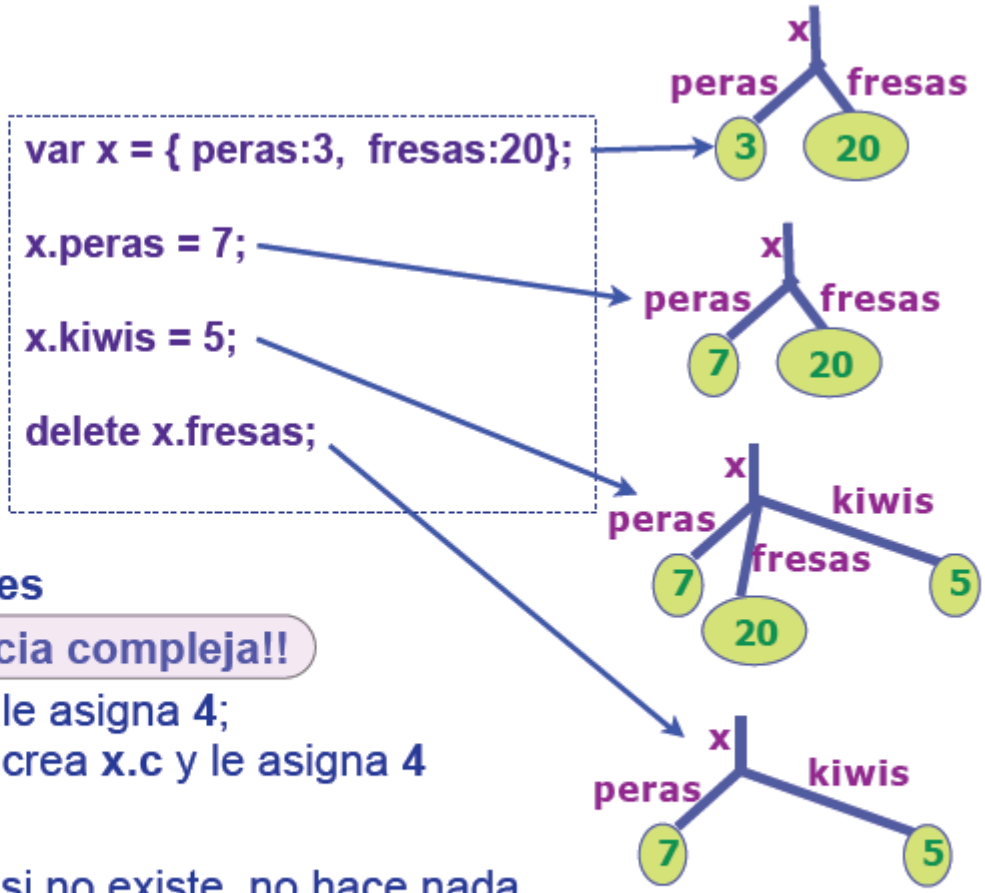
Actividad 7

- Utilizando JSON, definir la estructura de un objeto que almacena una factura. Las facturas están formadas por la información de la empresa (nombre de la empresa, dirección, teléfono, NIF), la información del cliente (similar a la de la empresa), una lista de elementos (cada uno de los cuales dispone de descripción, precio, cantidad) y otra información básica de la factura (importe total, tipo de IVA, forma de pago).
- Una vez definidas las propiedades del objeto, añadir un método que calcule el importe total de la factura y actualice el valor de la propiedad correspondiente. Por último añadir otro método que muestre por pantalla el importe total de la factura.

Objetos: características avanzadas

Propiedades dinámicas

- Las propiedades de objetos
  - Pueden **crearse**
  - Pueden **destruirse**
- Operaciones sobre **propiedades**
  - **x.c = 4** **¡¡OJO: sentencia compleja!!**
    - si propiedad **x.c** existe, le asigna **4**;
      si **x.c** no existe, crea **x.c** y le asigna **4**
  - **delete x.c**
    - si existe **x.c**, la elimina; si no existe, no hace nada
  - **"c" in x**
    - si **x.c** existe, devuelve **true**, sino devuelve, **false**

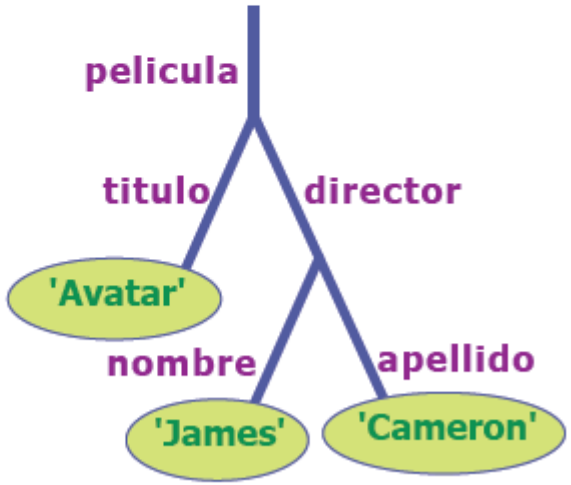


Objetos: características avanzadas

Objetos anidados: árboles

```
var pelicula = {
    titulo: 'Avatar',
    director:{
        nombre:'James',
        apellido: 'Cameron'
    }
};
```

- Los objetos pueden **anidarse** entre si
  - Los objetos anidados representan **arboles**



- La notación punto o array puede **encadenarse**
  - Representando un **camino en el árbol**
    - **Las siguientes expresiones se evaluan así:**
      - pelicula.director.nombre => 'James'
      - pelicula['director']['nombre'] => 'James'
      - pelicula['director'].apellido => 'Cameron'
      - pelicula.estreno => undefined
      - pelicula.estreno.año => Error_de_ejecución

Objetos: características avanzadas

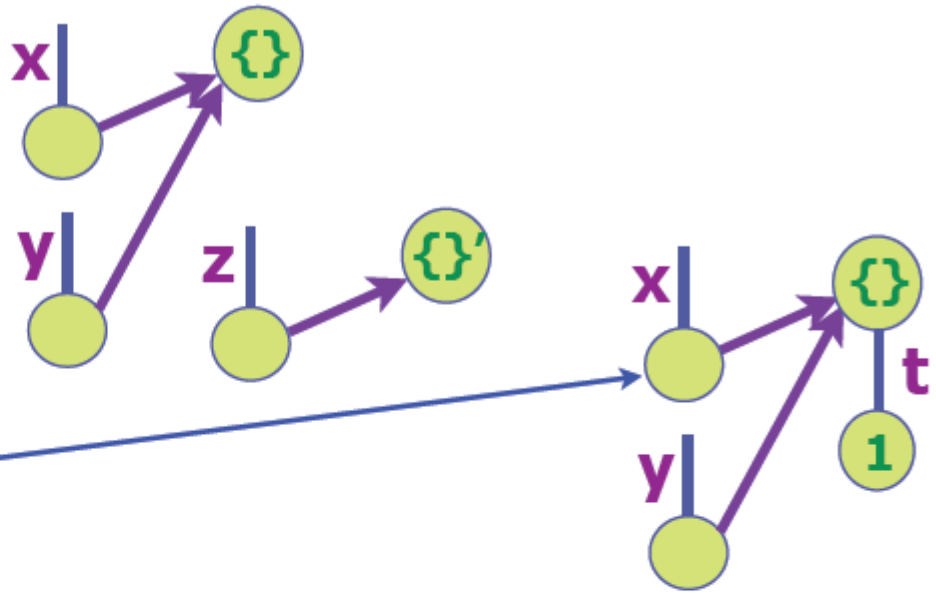
Usar propiedades dinámicas

- Las propiedades dinámicas de JavaScript
  - son muy útiles si se utilizan bien
- Un objeto solo debe definir las propiedades
  - que contengan información conocida
    - añadirá mas solo si son necesarias
- La información se puede consultar con
  - **prop1 && prop1.prop2**
    - para evitar errores de ejecución
      - si las propiedades no existen

```
// Dado un objeto pel definido con

var pel = {
      titulo: 'Avatar',
      director:'James Cameron'
};

// se puede añadir pel.estreno con

pel.estreno = {
    año:  '2009',
   cine: 'Tivoli'
}

// La expresión

pel.estreno && pel.estreno.año

// devuelve pel.estreno o undefined,
// evitando ErrorDeEjecución, si
// pel.estreno no se hubiese creado
```

Objetos: características avanzadas

```
var x = {};  // x e y tienen la
var y = x;   // misma referencia

var z = {};  // la referencia a z
             // es diferente de
             // la de x e y

x.t = 1;

x.t    => 1   // x accede al mismo
y.t    => 1   // objeto que y
z.t    => undefined
```



Referencias a objetos

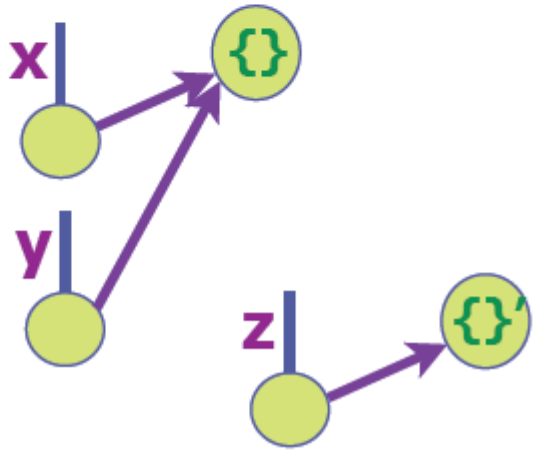
- Las variables que contienen objetos
  - solo contienen la referencia al objeto
- El objeto esta en otro lugar en memoria
  - indicado por la referencia
- Esto produce efectos laterales
  - como ilustra el ejemplo

Objetos: características avanzadas

Identidad de objetos

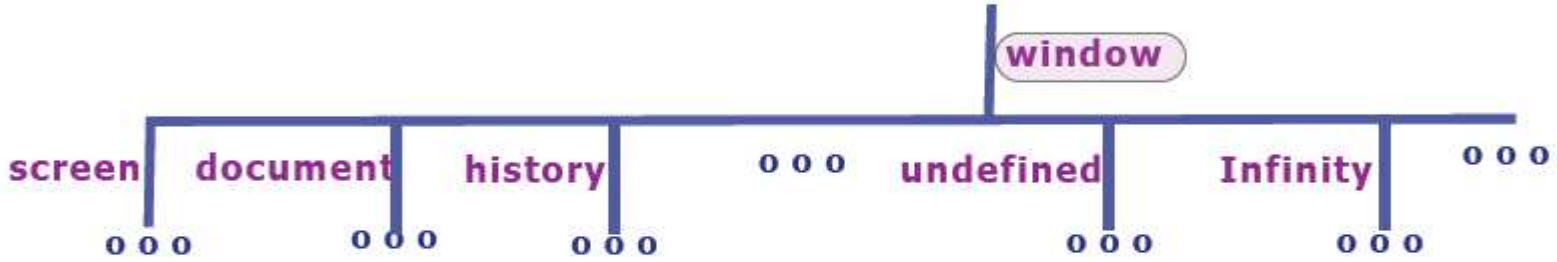
- Las referencias a objetos afectan a la identidad
  - porque identidad de objetos
    - es identidad de referencias
  - los objetos no se comparan
    - se comparan solo las referencias
  - es poco util si no se redefine
- Igualdad (debil) de objetos `==` y `!=`
  - no tiene utilidad tampoco con objetos
    - no se debe utilizar

```
var x = {};  // x e y contienen la
var y = x;   // misma referencia

var z = {}   // la referencia a z
             // es diferente de x e y

x === y          => true
x === {}         => false
x === z          => false
```



Objetos: características avanzadas

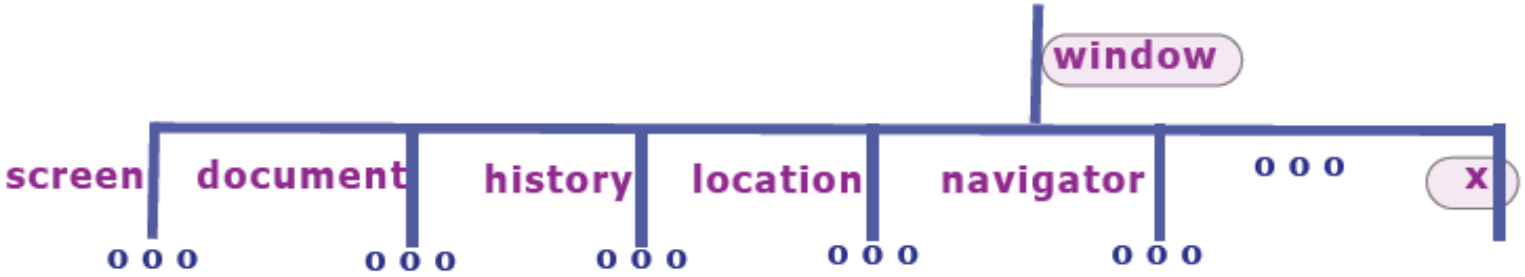
Objeto window o this



- El entorno de ejecución de JavaScript es el **objeto global window**
  - El **objeto global window** tiene propiedades con información sobre
    - **Objetos predefinidos** de JavaScript, el **navegador**, el **documento HTML**, ...
- **window** se referencia también como **this** en el entorno global
  - La **propiedad document** de **window** se referencia como
    - **window.document**, **this.document** o **document**
- Documentación: https://developer.mozilla.org/en-US/docs/Web/API/Window

Objetos: características avanzadas

Variables globales y el entorno de ejecución



- Un programa JavaScript se ejecuta con el objeto **window** como entorno
  - una asignación a una variable no definida como **x = 1;**
    - Crea una nueva **propiedad de window** de nombre **x**, porque
      - **x = 1;** es equivalente a **this.x = 1;** y a **window.x = 1;**
- Olvidar definir una variable, es un error muy habitual
  - y al asignar un valor a la variable no definida, JavaScript no da error
    - **sino que crea una nueva propiedad de window**
  - Es un error de diseño de JavaScript y hay que tratar de evitarlo

ANEXO sobre arrays

Métodos que emplean funciones

Hay métodos de arrays que pueden tener como parámetro una función para modificar los valores que formarán el resultado.

Los parámetros que se pueden pasar a la función, aunque no es obligatorio son:

1. `valor`: corresponde al valor de la posición actual del array.
2. `indice`: indica la posición del elemento dentro del array.
3. `arrayOrigen`: es el array completo con el que hemos comenzado a operar.

La función que declaremos no necesita tener definidos todos los parámetros, pero si deben estar en ese orden.

Función Callback

Es una función que se pasa como parámetro de otras funciones y esta función parámetro se ejecuta cuando la función principal ha terminado de ejecutarse, o cuando a nosotros nos interese.

En JavaScript son muy habituales, se utilizan en métodos, a los cuales se les pasa una función como parámetro.

Ejemplo:

```
function funcionPrincipal(callback)
{
 alert('hago algo y llamo al callback avisando que terminé');
 callback();
}

funcionPrincipal(function()
{
 alert('terminó de hacer algo');
});
```

Método forEach

Es un método que no devuelve nada y espera que se le pase una función como parámetro, que se ejecutará por cada elemento del array.

Sintaxis:

```
nomArray.forEach(fnTransformar)
```

A la función se le pueden pasar parámetros.

Este método es una forma de recorrer un array, se podría hacer lo mismo con un bucle for o while, pero de esta manera es mas elegante.

Método map

Es un método que recorre cada elemento del array y a través de una función callback que se le pasa como único parámetro, establece el calculo para cada elemento.

Este método NO modifica el array origen, sino que nos DEVUELVE otro array con los mismos elementos, al que se le ha aplicado la función.

Sintaxis:

```
nomArray.map(fnTransformar)
```

Método map

Ejemplo:

```
const notas = [5,6,,,9,,8,7,8,9,,10];
const notasConPeso= notas.map(x=> x*1,2);
console.log(notasConPeso);
```

El nuevo array tendría las notas actualizadas al 20%.

Lo hemos hecho con la función flecha, pero se podría hacer declarando la función:

```
function SumarPeso(valor){
    return valor*1,2;
}
const notasConPeso= notas.map(sumarPeso);
console.log(notasConPeso);
```

Función flecha

Es una función anónima, relacionada con métodos y no pueden ser usada como constructor.

Se definen utilizando una flecha =>

Sintaxis:

```
(param1, param2, …rest) => {expresión;}
```

Ejemplo:

var suma = (x, y) => x+y;

Método reduce

Es un método aplica la función callback a cada elemento del array para ir formando o acumulando un resultado, reduciendo así el array a un único valor de salida que puede ser de cualquier tipo.

Este método devuelve un valor.

Sintaxis:

```
nomArray.reduce (fnAcumular, valor)
```

Ejemplo:

```
const array=[1,2,3,4,5];
let suma=array.reduce((acu,valor)=>acu+valor,0);
console.log(suma);
```

Método filter

Genera un array con los elementos que cumplen con la función callback fnComprobar. Si ninguno cuadra con las condiciones de la función, devuelve el array vacio..

Este método devuelve un array con los elementos que cumplen la condición de la función.

Sintaxis:

```
nomArray.filter(fnComprobar)
```

Ejemplo:

```
const array=[4,9,2,6,5,7,8,1,10,3];
const arrayFiltrado=array.filter(x=>x>5);
console.log(arrrayFiltrado)
```