

Proyecto BID-MINEC

UR PROYECTO UPSKILLING & RESKILLING

Orientado a la reconversión y mejora de
habilidades digitales y tecnológicas



FrontEnd básico-intermedio con JavaScript

Introducción

Facilitador: Iván Alvarado



JavaScript

Contenido

1. Introducción
2. Variables
3. Operadores
4. Estructuras de control
5. Funciones
6. Arreglos
7. Document Object Model

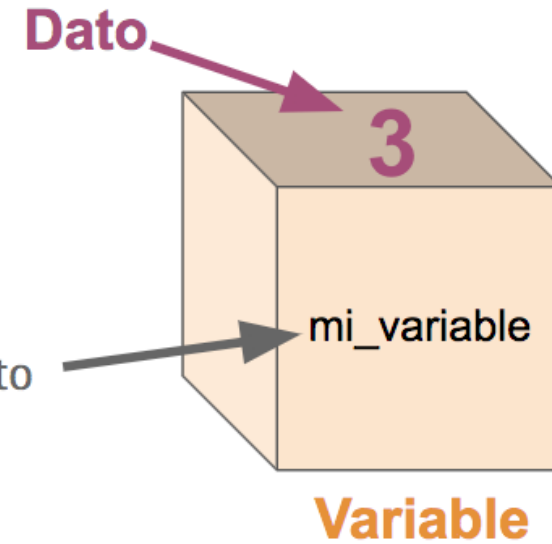
1. Introducción, usos de JavaScript

- JavaScript es un lenguaje que se ejecuta del lado del cliente y es interpretado por los browsers.
- La primera versión de JavaScript se liberó en diciembre de 1995.
- La versión actual de JavaScript es 1.7. (Se puede revisar en este link <http://jsfiddle.net/Ac6CT/>)
- Su nombre estandarizado es ECMAScript, pero la mayoría de la gente lo conoce como JavaScript.
- JavaScript fue desarrollado por Netscape.
- Jscript es un lenguaje desarrollado por Microsoft y es muy parecido a JavaScript.
- Las últimas versiones de JScript y JavaScript cumplen con las especificaciones de ECMAScript

2. Variables

Variables JS

Cada variable tiene un nombre, de modo que podamos acceder a ese dato siempre que necesitemos.



variable identifier

start with

assignment operator

value

End of the statement

```
var name = 'James Bond';
```

3. Operadores

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Substracción	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo: el resto después de la división	7 % 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	-a	Invierte el signo de un operando.

3. Operadores

Operadores lógicos

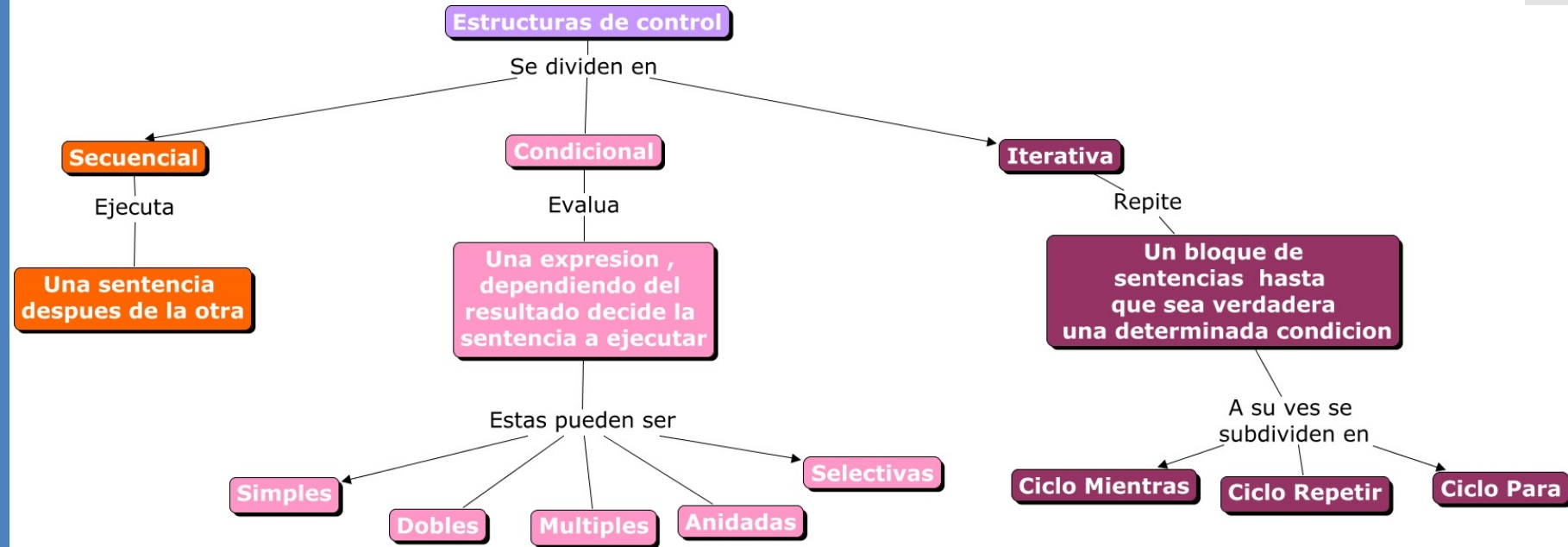
Operador	Uso	Descripción
AND Lógico (&&)	<code>expr1</code> && <code>expr2</code>	Devuelve <code>expr1</code> si puede ser convertido a <code>false</code> de lo contrario devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, && devuelve <code>true</code> si ambos operandos son <code>true</code> , en caso contrario devuelve <code>false</code> .
OR Lógico ()	<code>expr1</code> <code>expr2</code>	Devuelve <code>expr1</code> si puede ser convertido a <code>true</code> de lo contrario devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, devuelve <code>true</code> si alguno de los operandos es <code>true</code> , o <code>false</code> si ambos son <code>false</code> .
NOT Lógico (!)	<code>!expr</code>	Devuelve <code>false</code> si su operando puede ser convertido a <code>true</code> , en caso contrario, devuelve <code>true</code> .

3. Operadores




Operadores con Asignación

Operador	Operación	Valor x	Valor y	Resultado	Equivale a
=	$x = y$	10	5	$x = 5$	
+=	$x += y$	10	5	$x = 15$	$x = x + y$
-=	$x -= y$	10	5	$x = 5$	$x = x - y$
*=	$x *= y$	10	5	$x = 50$	$x = x * y$
/=	$x /= y$	10	5	$x = 2$	$x = x / y$
%=	$x \% =$	10	5	$x = 0$	$x = x \% y$

4. Estructuras de control



5. Funciones

```
      NAME       PARAMETERS   
function addNumbers(a, b) {  
  BODY     return a + b;  
}
```

5. Funciones

```
function Suma(num1, num2)
{
    var n;
    n = num1 + num2;

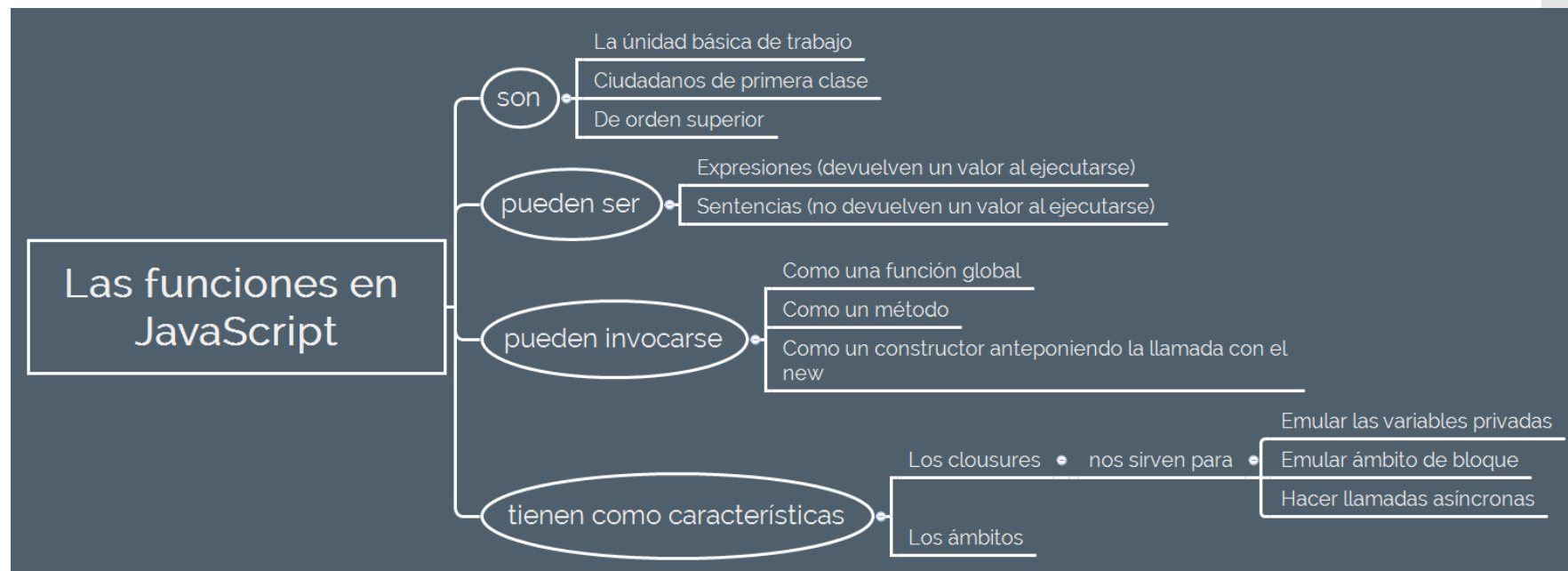
    return n;
}

var resultado;

resultado = Suma (3, 5);

document.write('La suma es: ', resultado);
```

5. Funciones



5. Funciones

Funciones de primera clase.

En un lenguaje de programación se dice que tiene **Funciones de primera clase** cuando las funciones en ese lenguaje son tratadas como cualquier otra variable.

Por ejemplo, en ese lenguaje:

- Puede ser asignada a una variable
- Una función puede ser pasada como argumento a otras funciones
- Puede ser retornada por otra función .

Ejemplo | Asignar función a una variable

JavaScript

```
const foo = function() {  
    console.log("foobar");  
}  
// Invocación usando una variable  
foo();
```

5. Funciones

Ejemplo | Pasar la función como argumento

JavaScript

```
function diHola() {  
    return "Hola ";  
}  
  
function saludar(saludo, nombre) {  
    console.log(saludo() + nombre);  
}  
  
// Pasamos `diHola` como argumento de la función `saludar`  
saludar(diHola, "JavaScript!");
```

Ejemplo | Devolver una función

JavaScript

```
function diHola() {  
    return function() {  
        console.log("¡Hola!");  
    }  
}
```

En este caso también la llamada puede ser con doble paréntesis

5. Funciones

Funciones de orden superior.

Funciones que llaman a otras funciones o que devuelven funciones (closures), se conocen como funciones de orden superior.

Sirven para esconder el detalle, es decir, proporcionan un mayor nivel de abstracción, permitiéndonos pensar a un mayor nivel de abstracción.

Ejemplos típicos (NaN Not-A-Number)

forEach, filter, map, reduce, every, some... (if [NaN,NaN,NaN].every(isNaN){...}).

Se aplican : Composición(las funciones pueden componerse), Recursión(Pueden llamarse a sí mismas) y Binding(Todas las funciones tienen el metodo bind, ello permite crear una nueva función fijando alguno de los parámetros).

Ejemplo bind

```
function f(a,b){ return a+b}
```

```
var x = f.bind(a, 2);
```

5. Funciones

Closures

Un closure es cuando una función es capaz de recordar y acceder a un lexical scope, incluso cuando la función es ejecutada por fuera del lexical scope.

Los closures están disponibles gracias a que el lenguaje implementa lambdas y funciones de orden superior y son una consecuencia directa de escribir código usando lexical scopes.

Dado el siguiente ejemplo:

```
function Principal(){  
    var VariableInterna="Cadena";  
    function Secundaria(){  
        document.write(VariableInterna);  
    }  
    return Secundaria;  
}  
  
var ejemplo = Principal();  
Ejemplo();
```

Al ejecutar la función Principal(), cuando se asigna la variable. ¿esta no debería ser borrada por el GarbageCollector, ya que no está en uso?.

Esto no ocurre, porque se retorna la definición de la función interna que tiene una referencia a este scope.

Lo fundamental para entender el concepto de closure, es que se cuenta con un contexto y una función que hace uso de este, permitiendo acceder al scope de tal contexto.

6. Arreglos

ArrayBuffer (16 bytes)

UInt8Array	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UInt16Array	0		1		2		3		4		5		6		7	
UInt32Array	0				1				2				3			
Float64Array	0								1							

6. Arreglos

```
var vector = new Array(3);
```

```
vector[0] = 3;
```

```
vector[1] = 1;
```

```
vector[2] = 7;
```

```
document.write(vector[0], '<br>');
```

```
document.write(vector[1], '<br>');
```

```
document.write(vector[2], '<br>');
```

```
document.write('La longitud del array es: ', vector.length);
```

6. Arreglos

Aplicación de foreach en arreglos

- Implementando callback con funciones anónimas

Ejemplo:

```
var arreglo = [1,2,3,4];  
arreglo.forEach(function(elemento, índice, arreglo)){  
    document.write(elemento);  
}
```

6. Arreglos

Aplicación de foreach en arreglos

- Expresiones lambda

Ejemplo:

```
var arreglo = [1,2,3,4];  
arreglo.forEach( elemento =>{  
    document.write(elemento);  
});
```

- Implementando “in” o “of” al recorrer el arreglo

6. Arreglos

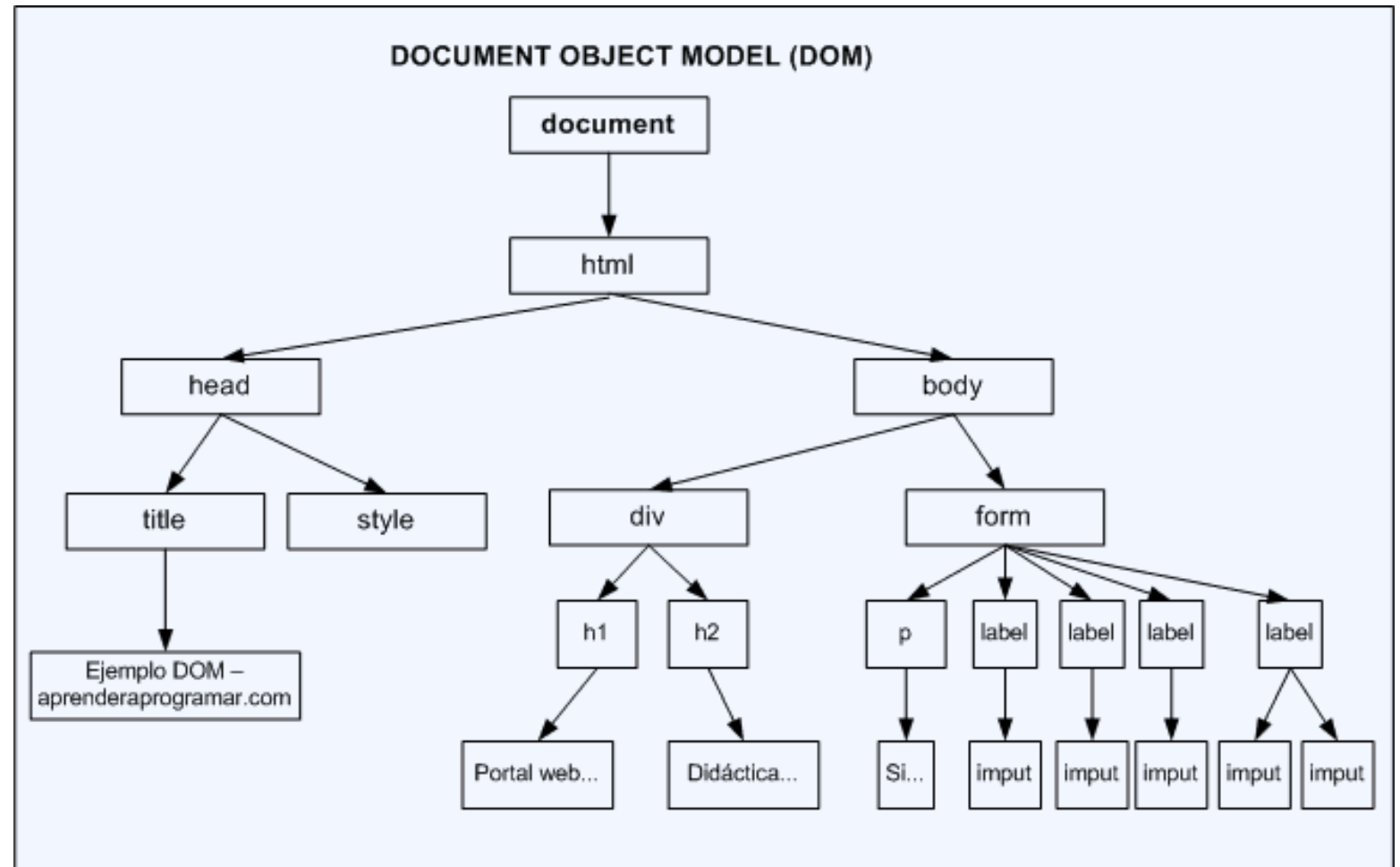
Aplicación de foreach en arreglos

- Implementando “in” o “of” al recorrer el arreglo

Ejemplo:

```
var arreglo = [1,2,3,4];  
for (var e in arreglo){  
    document.write(arreglo[e]);  
}  
  
for (var e of arreglo){  
    document.write(e);  
}
```

7. Document Object Model



7. Document Object Model

- Qué es el DOM

El modelo de objeto de documento (DOM) es una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su estructura, estilo y contenido.

El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación.

Una página web es un documento. Éste documento puede exhibirse en la ventana de un navegador o también como código fuente HTML. Pero, en los dos casos, es el mismo documento. El modelo de objeto de documento (DOM) proporciona otras formas de presentar, guardar y manipular este mismo documento. El DOM es una representación completamente orientada al objeto de la página web y puede ser modificado con un lenguaje de script como JavaScript.

7. Document Object Model

- El DOM no es un lenguaje de programación pero sin él, el lenguaje JavaScript no tiene ningún modelo o noción de las páginas web, de la páginas XML ni de los elementos con los cuales es usualmente relacionado.
- Cada elemento -"el documento íntegro, el título, las tablas dentro del documento, los títulos de las tablas, el texto dentro de las celdas de las tablas"- es parte del modelo de objeto del documento para cada documento, así se puede acceder y manipularlos utilizando el DOM y un lenguaje de escritura, como JavaScript.
- En el comienzo, JavaScript y el DOM estaban herméticamente enlazados, pero después se desarrollaron como entidades separadas.
- El contenido de la página es almacenado en DOM y el acceso y la manipulación se hace vía JavaScript, podría representarse aproximadamente así:

7. Document Object Model

API (web o página XML) = DOM + JS (lenguaje de script)

- El DOM fue diseñado para ser independiente de cualquier lenguaje de programación particular, hace que la presentación estructural del documento sea disponible desde un simple y consistente API.
- No se tiene que hacer nada especial para empezar a utilizar el DOM. Los diferentes navegadores tienen directrices DOM distintas, y éstas directrices tienen diversos grados de conformidad al actual estándar DOM (un tema que se intenta evitar en esta presentación), pero todos los navegadores web usan el modelo de objeto de documento para hacer accesibles las páginas web al script.

7. Document Object Model

Tipos de datos

Esta parte intenta describir, de la manera más simple posible, los diferentes objetos y tipos. Pero hay que conocer una cantidad de tipos de datos diferentes que son utilizados por el API. Para simplificarlo, los ejemplos de sintaxis en esta API se refieren a:

- nodos como elements
- a una lista de nodos como nodeLists (o simples elementos)
- a nodos de atributo como attributes.

document

Cuando un miembro devuelve un objeto del tipo document (por ejemplo, la propiedad **ownerDocument** de un elemento devuelve el documento "document" al cual pertenece), este objeto es la raíz del objeto documento en sí mismo.

element

element se refiere a un elemento o a un nodo de tipo de elemento "element" devuelto por un miembro del API de DOM. Dicho de otra manera, por ejemplo, el método document.createElement() devuelve un objeto referido a un nodo, lo que significa que este método devuelve el elemento que acaba de ser creado en el DOM. Los objetos element ponen en funcionamiento a la interfaz Element del DOM y también a la interfaz de nodo "Node" más básica, las cuales son incluidas en esta referencia.

7. Document Object Model

- Tipos de datos

nodeList

Una "nodeList" es una serie de elementos, parecido a lo que devuelve el método `document.getElementsByTagName()`. Se accede a los items de la nodeList de cualquiera de las siguientes dos formas:

`list.item (1)`

`lista [1]`

Ambas maneras son equivalentes. En la primera, **item()** es el método del objeto nodeList. En la última se utiliza la típica sintaxis de acceso a listas para llegar al segundo ítem de la lista.

attribute

Cuando un atributo ("attribute") es devuelto por un miembro (por ej., por el método **createAttribute()**), es una referencia a un objeto que expone una interfaz particular (aunque limitada) a los atributos. Los atributos son nodos en el DOM igual que los elementos, pero no suelen usarse así.

NamedNodeMap

Un namedNodeMap es una serie, pero los ítems son accesibles tanto por el nombre o por un índice, este último caso es meramente una conveniencia para enumerar ya que no están en ningún orden en particular en la lista. Un NamedNodeMap es un método de ítem() por esa razón, y permite poner o quitar ítems en un NamedNodeMap

7. Document Object Model

- Element o mejor conocido como Nodo

Metodos del document que devuelven un Nodo (Element)

- Método createElement();
- Método getElementById();

Ejemplos:

```
var encabezado = document.createElement("h1");  
var cont = document.createTextNode("Algo");  
document.write(encabezado.nodeType); //1, un ELEMENT_NODE  
document.write(cont.nodeType); //3, un TEXT_NODE
```

7. Document Object Model

Codigos devueltos por el nodeType del Element (Nodo)

<u>ELEMENT_NODE</u>	1
ATTRIBUTE_NODE	2
<u>TEXT_NODE</u>	3
CDATA_SECTION_NODE	4
ENTITY_REFERENCE_NODE	5
ENTITY_NODE	6
<u>PROCESSING_INSTRUCTION_NODE</u>	7
<u>COMMENT_NODE</u>	8
<u>DOCUMENT_NODE</u>	9
<u>DOCUMENT_TYPE_NODE</u>	10
<u>DOCUMENT_FRAGMENT_NODE</u>	11
NOTATION_NODE	12

7. Document Object Model

Codigos devueltos por el nodeValue del Element (Nodo)

Para el nodo en si, nodeValue devuelve null. Para texto, comentarios, y nodos CDATA, nodeValue devuelve el contenido de dicho nodo. Para nodos atributos, el valor del atributo es devuelto.

Atributo	valor del atributo
CDATASection	contenido de la sección CDATA
Comment	Contenido del comentario
Document	null
DocumentFragment	null
DocumentType	null
Element	null
NamedNodeMap	null
EntityReference	null
Notation	null
ProcessingInstruction	Todo el contexto excluyendo el objetivo
Text	Contenido del nodo de texto

7. Document Object Model

- Propiedades del Element (Nodo)

Propiedades

childNodes (Arreglo de elementos)
firstChild
isConnected
lastChild
nextSibling
nodeName (Nombre de etiqueta)
nodeType (Tipo de elemento)
nodeValue (Cadena de nodo txt)
ownerDocument
parentElement
parentNode
previousSibling
textContent

Métodos

appendChild()
cloneNode()
compareDocumentPosition()
contains()
getRootNode()
hasChildNodes()
insertBefore()
isDefaultNamespace()
isEqualNode()
lookupNamespaceURI()
lookupPrefix()
normalize()
removeChild()
replaceChild()

7. Document Object Model

- Ejemplo de uso de childNodes

```
var ob = document.getElementById("Id1");
var x = ob.childNodes;
for (var i=0; i< x.length; i++){
    if (x.item(i).nodeName != "#text") // x[i];
        alert(x.item(i).innerHTML);
}
```

HTML

```
<div id="Id1">
    <p>Primer parrafo</p>
    texto
    <p>segundo parrafo</p>
</div>
```


7. Document Object Model

Diferencia en el uso nodeValue y textContent

```
var ob = document.getElementById("Id1");  
document.write(ob.nodeValue); //Muestra null, porque es el element  
document.write (ob.textContent); //Muestra completo
```

HTML

```
<div id="Id1">  
  <p>Primer parrafo</p>  
  texto  
  <p>segundo parrafo</p>  
</div>
```

7. Document Object Model

Alterando estilos

```
var encabezado = document.createElement("h1");  
var texto = document.createTextNode("Titulo");  
encabezado.style.color = "red";  
encabezado.style.fontSize = "50px";  
  
encabezado.appendChild(texto);  
document.body.appendChild(encabezado);
```

7. Document Object Model

Inputs:

- email (Para capturar dato de tipo correo electrónico)
- search (Muy parecido al text, salvo que guarda valores y reutiliza en múltiples páginas, además que agrega el efecto visual para borrar lo digitado).
- tel (Para especificar datos numéricos de un contacto telefónico, útil para dispositivos táctiles)
- url (Útil para direcciones url)
- number (datos numéricos con scroll de incremento/decremento)
- range (Es un dato numérico pero con forma de slider)
- date, time, datetime-local (Presenta todo un calendario)
- color (Presenta una paleta de colores)