Javascript

JavaScript

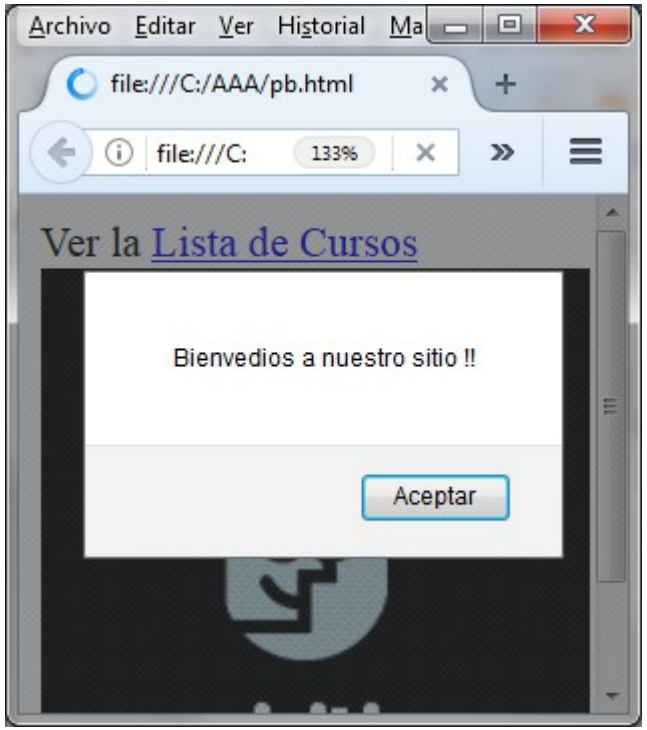
- Es un lenguaje diferente a JAVA
- Es solamente orientado a Web
- JavaScript es usualmente interpretado y no compilado
- Es soportado por los browsers más comunes
- El código JavaScript se incrusta en una página HTML usando la etiqueta ***script***

```
<html>
<body>
  Ver la <a href ="Lista.html"> Lista de Cursos</a>
  <img src="misitio.gif">
  <script language="javascript">
      window.alert("Bienvedios a nuestro sitio");
  </script>
</body>
</html>
```

JavaScript : elementos

- Dentro de un script se pueden incluir:
  - Variables (pueden referir datos, objetos y funciones)
  - Funciones
  - Instrucciones (las que estén directo en el script se ejecutan de una vez)

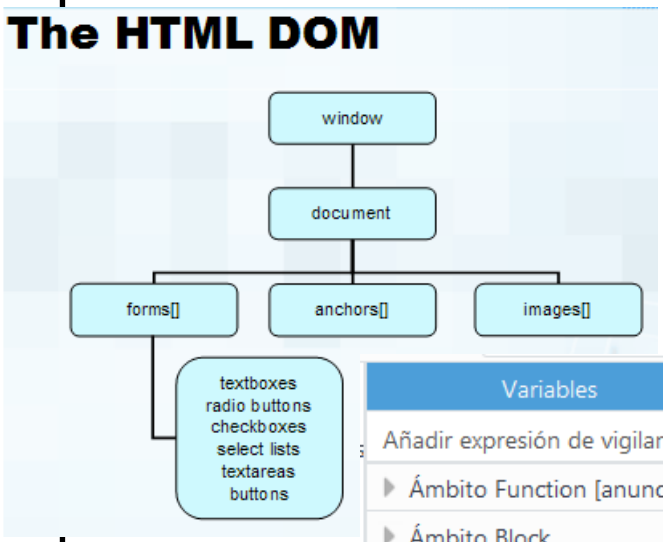
```
<html>
<body>
 Ver la <a href ="Lista.html"> Lista de Cursos</a>
 <img src="misitio.gif">
 <script language="javascript">
     var anuncio;
      function anunciar(){
            var simbolo = " !!";
            window.alert(anuncio+simbolo);
      };
      anuncio="Bienvedios a nuestro sitio";
      anunciar();
 </script>
</body>
</html>
```

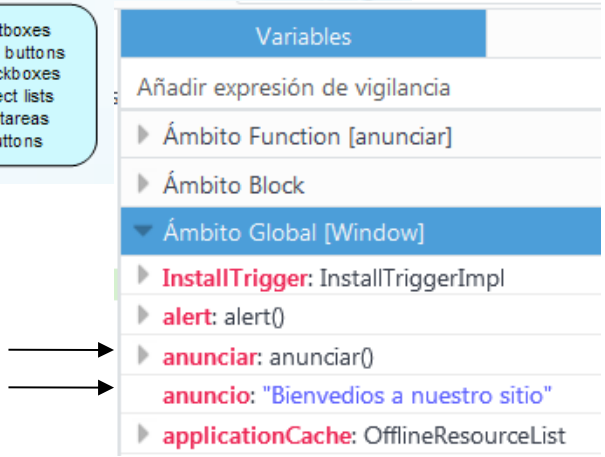


JavaScript : ámbito (*scope*) de los elementos

- En HTML, Los elementos (variables y funciones) declarados ***directamente*** en un script son puestos en el ámbito global, que corresponde al objeto "*window*" creado por el browser.

```
<html>
<body>
 Ver la <a href ="Lista.html"> Lista de Cursos</a>
 <img src="misitio.gif">
 <script language="javascript">
   var anuncio;
   function anunciar(){
     var simbolo = " !!";
     window.alert(window.anuncio+simbolo);
   };
   anuncio="Bienvedios a nuestro sitio";
   window.anunciar();
 </script>
</body>
</html>
```

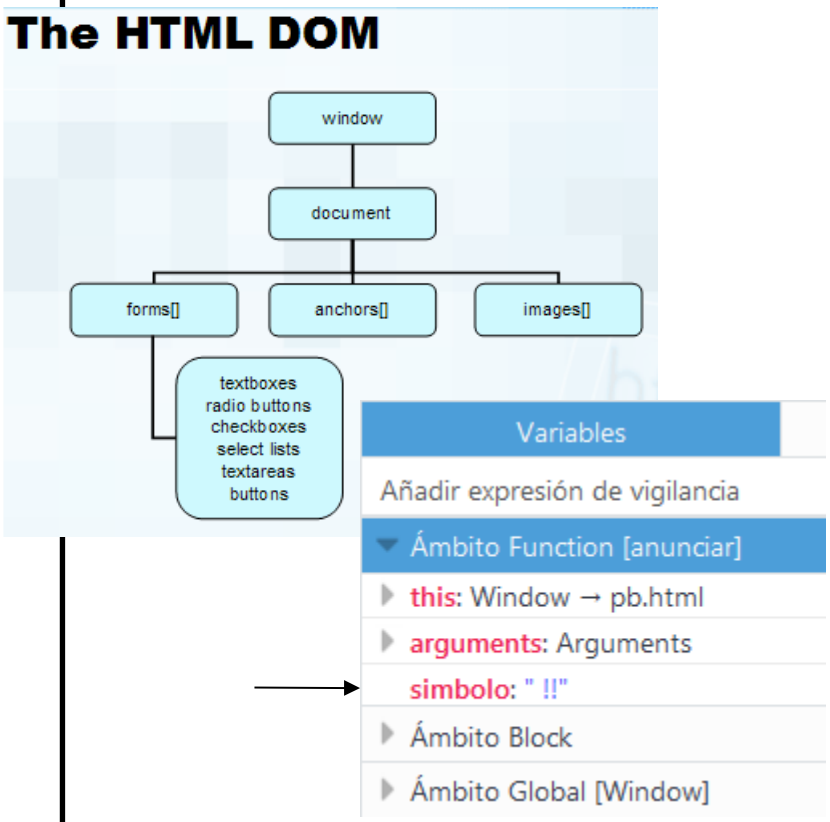




JavaScript : ámbito (*scope*) de los elementos

- Los elementos declarados con ***var*** dentro de una función quedan en el ambito de esa función.
- Los elementos introducidos sin var, quedan en el ámbito global

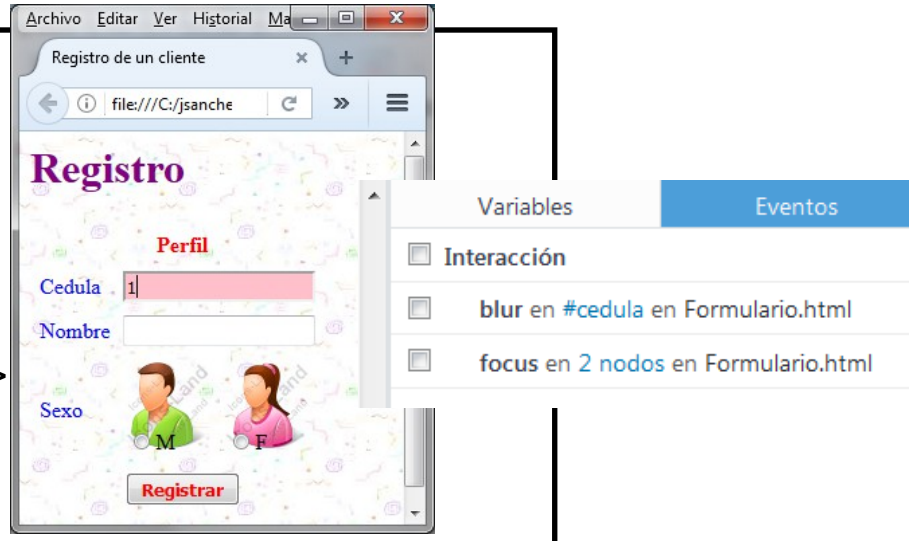
```
<html>
<body>
 Ver la <a href ="Lista.html"> Lista de Cursos</a>
 <img src="misitio.gif">
 <script language="javascript">
   var anuncio;
   function anunciar(){
     var simbolo = " !!";
     window.alert(window.anuncio+simbolo);
   };
   anuncio="Bienvedios a nuestro sitio";
   window.anunciar();
 </script>
</body>
</html>
```



Eventos

- Los elementros HTML generan eventos (*click*, *changed*, etc.)
- Cada elemento HTML (su objeto DOM correspondiente) tiene atributos donde “almacena” los event-handlers que se ejecutarán en respuesta a los eventos respectivos.
- Esos event-handlers pueden asociarse al elemento en el HTML o por programación.

```
...
<input type="text" name="cedula" id="cedula"
    onfocus="this.className='focus';"
    onblur="this.className='nofocus';">
...
<input type="text" name="nombre" id="nombre" >
…
<script>
  function doFocus(event){ event.target.className="focus";  }
   function doBlur(event){  event.target.className="nofocus"; }
   document.getElementById("nombre").addEventListener("focus",doFocus);
   document.getElementById("nombre").addEventListener("blur",doBlur);
</script>
```



Evento *DOMContentLoaded*

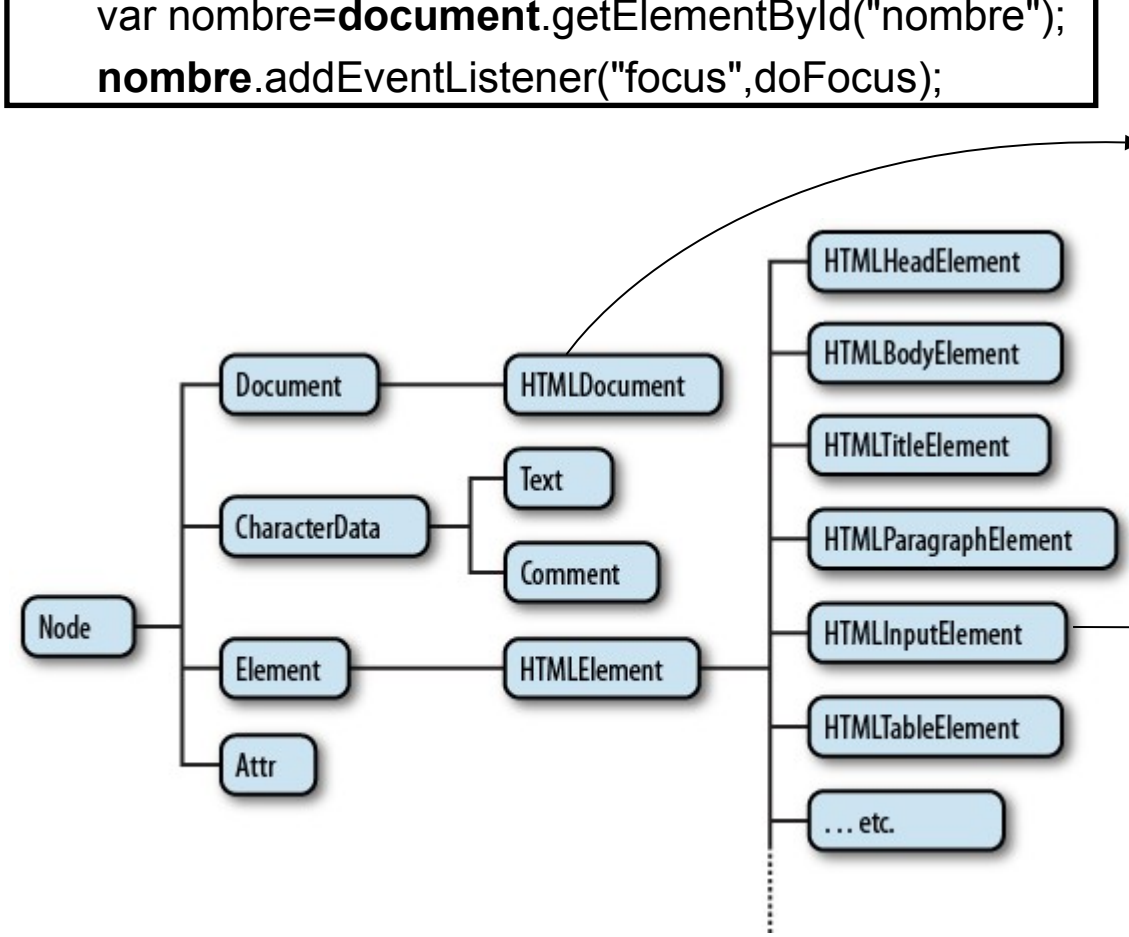
- El browser construye el árbol de objetos DOM desde el HTML
- El código JavaScript que manipule los objeto del DOM (*document* y sus descendientes) no debería ejecutarse sino hasta que el árbol DOM completo haya sido creado.
- El evento *DOMContentLoaded* se dispara inmediatamente después de que el árbol DOM es construido.
- Puede asociársele un event-handler que implemente cualquier procesamiento requerido cuando la página es cargada.

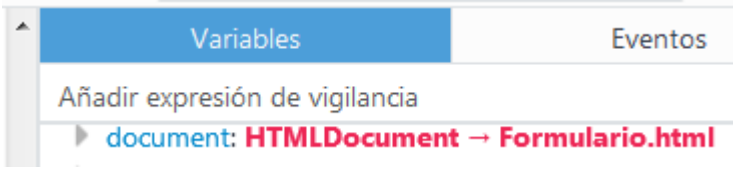
```
<script>
  function pageLoad(event){
    var nombre=document.getElementById("nombre");
    nombre.addEventListener("focus",doFocus);
    nombre.addEventListener("blur",doBlur);
  }
  function doFocus(event){ event.target.className="focus";  }
  function doBlur(event){ event.target.className="nofocus"; }
  document.addEventListener("DOMContentLoaded", pageLoad)
</script>
```

Interfaces DOM (Document Object Model)

- Los nodos del árbol construido por el browser implementan las interfaces definidas en la especificación DOM

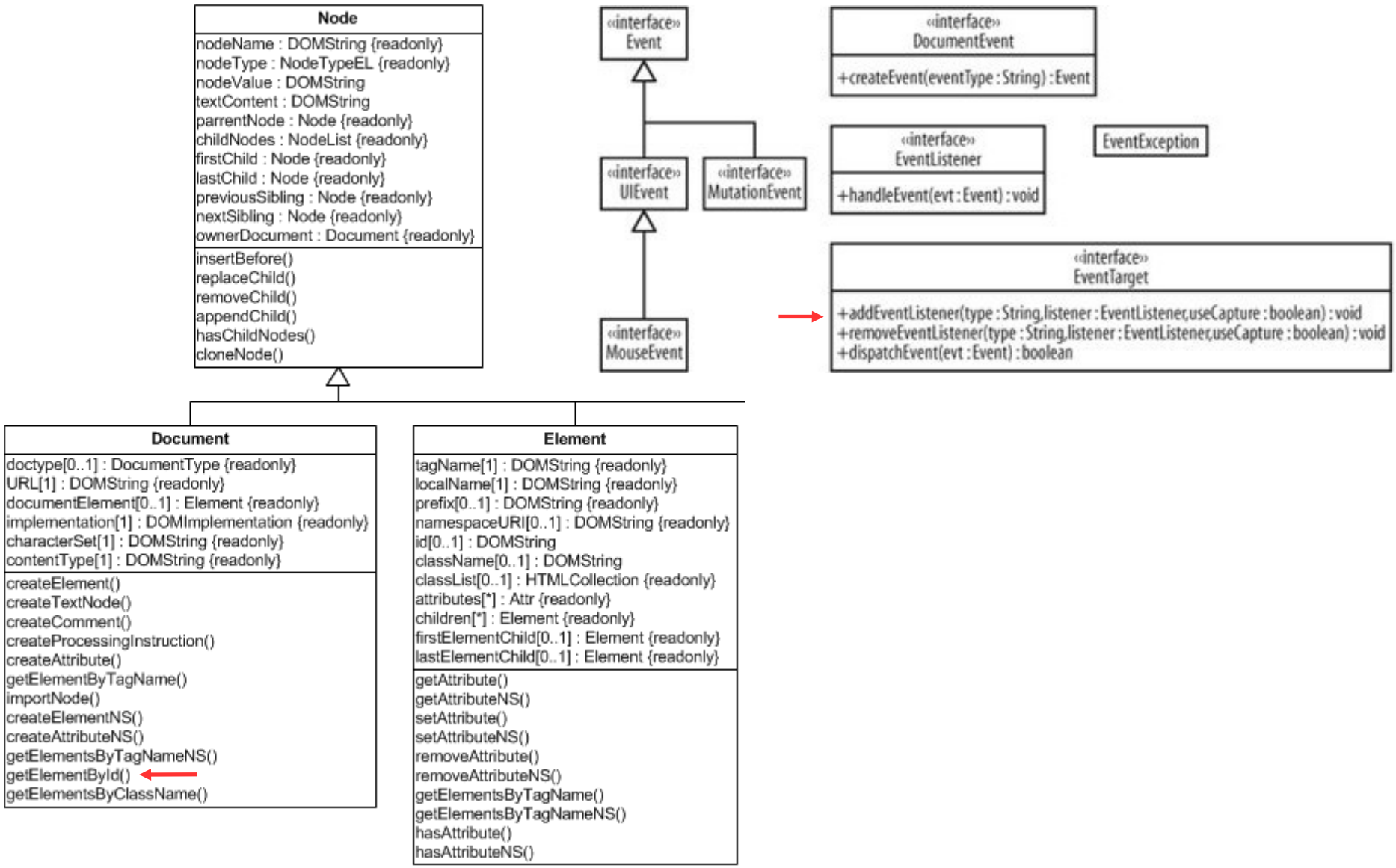
```
function pageLoad(event){
  var nombre=document.getElementById("nombre");
  nombre.addEventListener("focus",doFocus);
```





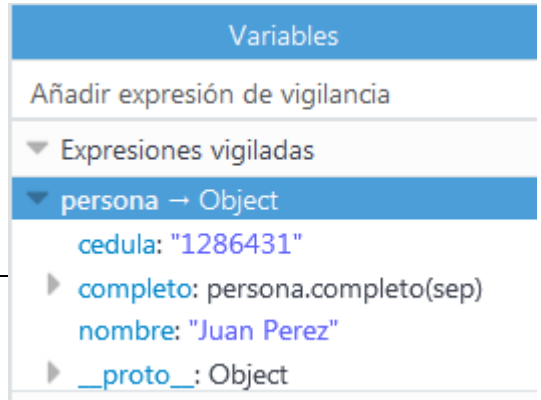
Interfaces DOM (Document Object Model)

```
var nombre=document.getElementById("nombre");
nombre.addEventListener("focus",doFocus);
```



Objetos

- Un objeto es una coleccion de atributos, cada uno con su valor.
- Los atributos pueden tener datos o funciones
- Los objetos se pueden “construir” de varias meneras
  - **literales**
  - con *CreateObject*
  - con *new*




```
persona = {
    cedula: "1286431",
    nombre: "Juan Perez",
   completo: function (sep) { return this.cedula +  sep + this.nombre; }
};
console.log(persona.completo("-"));
```

Prototype

- Un Prototype es un objeto
- Todo objeto tiene un prototype del cual “hereda”
  - Es el eslabón “padre” en la cadena de búsqueda del objeto
  - Se identifica como _proto_ en cada objeto

```
persona = {
   cedula: "1286431",
   nombre: "Juan Perez",
   completo: function (sep) {
         return this.cedula +  sep + this.nombre;
       }
 };
 personaNacional = Object.create(persona);
 personaNacional.pais="Costa Rica";
 console.log(personaNacional.completo("-"));
 console.log(personaNacional.toSource());
```

Creacion de Objetos con *NEW*

- NEW se usa con una función: new f(p)
- NEW crea el objeto e invoca la función (f) para que lo inicialice

```
function f (cedula,nombre){
  this.cedula=cedula;
  this.nombre=nombre;
  this.completo= function (sep) { return this.cedula +  sep + this.nombre; };
}
persona = new f("1286431", "Juan Perez");
console.log(persona.completo("-"));
```

NEW con Prototype

- Toda función tiene una propiedad llamada *prototype* que almacena un objeto
- Todos los objetos creados invocando **new** con una función toman como prototype (_proto_) el objeto que esté en la propiedad *prototype* de esa función.
- Se puede asociar una objeto nuestro a la propiedad prototype de una función
- Los objetos creados con **new** sobre esa función compartirán ese objeto prototype
- Ese objeto prototype tendría todas las funciones (métodos) de nuestra “clase”.
- Cada objeto creado con la función tendrá su juego de variables

NEW con Prototype


```
// funcion Persona
function Persona (cedula,nombre){
  this.init(cedula,nombre);

// prototype asociado a la funcion Persona
// solo metodos
Persona.prototype={
    init: function(cedula,nombre){
       this.cedula=cedula;
       this.nombre=nombre;
    },
   completo: function (sep) { return this.cedula +  sep + this.nombre; }
}
persona = new Persona("1286431", "Juan Perez");
console.log(persona.completo("-"));
```

Storage

- Los browser proveen objetos donde el programa Javascript puede almacenar y recuperar datos en la máquina del cliente.
- Los *valores* que se pueden alacenar son Strings y son accesibles por medio de un *llave* de tipo String
  - setItem(key,value)
  - getItem(key)->value
- ***sessionStorage***: peramanece, y conserva su contenido, por la duración de la sesión (ventana/pestaña se mantine abierta)
- ***localStorage***: permanece, y conserva su contenido, permanentemente, para cada dominio.

JSON (JavaScript Object Notation)

- Es un formato sencillo para intercambio de datos
- Es un formato de **texto**
- Es independiente de todo lenguaje de programación
- Existen parsers para convertir de JSON a distintos lenguajes de programación y de esos lenguajes a JSON.
  - JavaScript ⟺ JSON (parser objeto **JSON** de JavaScript)
    - stringify: JavaScript ⟹ JSON
    - parse: JSON ⟹ JavaScript
  - Java ⟺ JSON (clase **Gson** en Java)
    - toJson: Java ⟹ JSON
    - fromJson: JSON ⟹ Java
  - …