

PROYECTO DE SOFTWARE

¿QUÉ ABORDAREMOS EN ESTE VIDEO?

- Una introducción al lenguaje JavaScript
- DOM: una descripción más detallada.

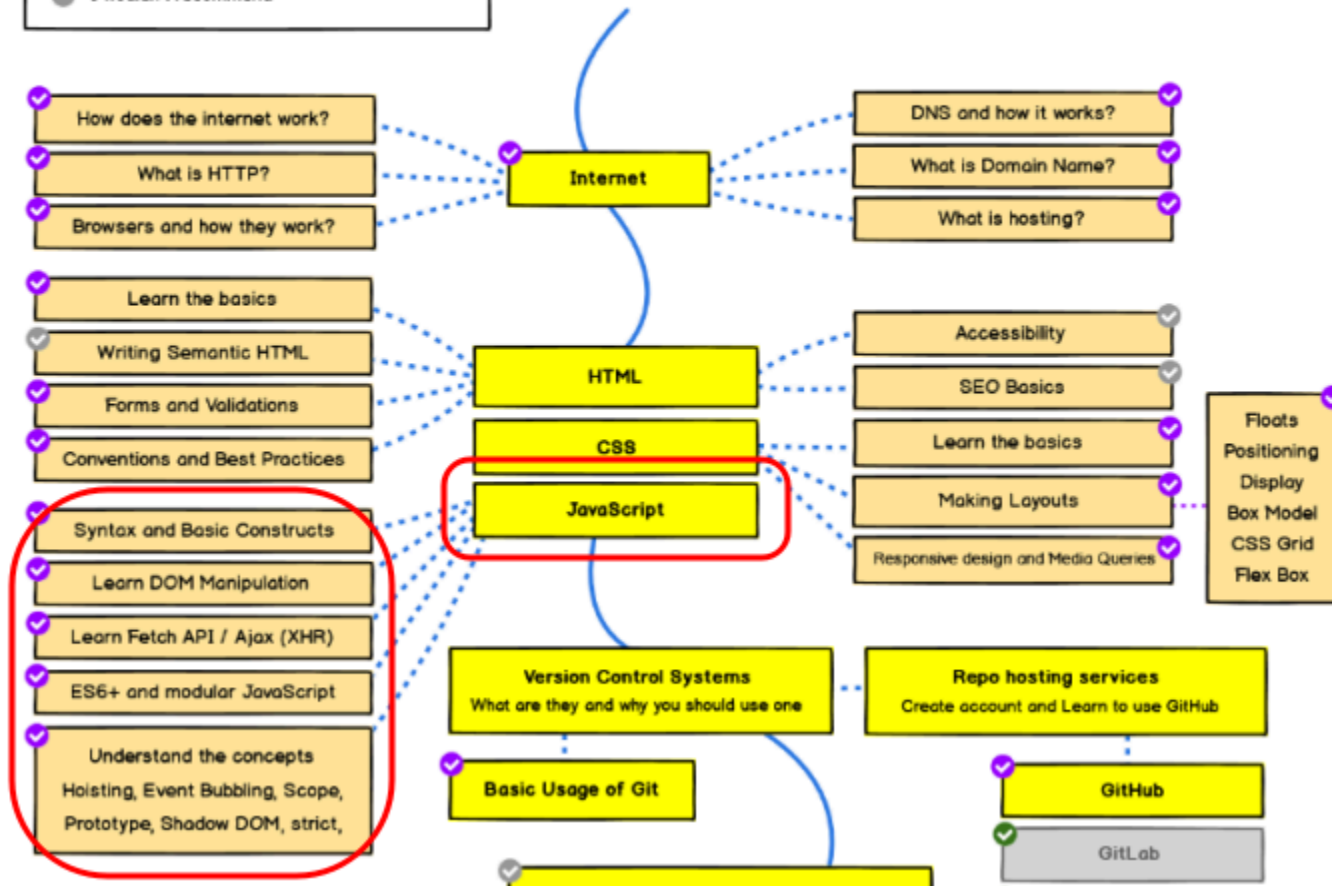
PROCESAMIENTO EN EL CLIENTE

- ✓ Personal Recommendation / Opinion
- ✓ Alternative Option - Pick this or purple
- ✓ Order in roadmap not strict (Learn anytime)
- I wouldn't recommend

Find the detailed version of this roadmap along with resources and other roadmaps

<http://roadmap.sh>

Front-end



¿QUÉ SABEMOS DE LOS SCRIPTS?

- Los scripts son pequeños programas que se incluyen en el documento html.
- Si el navegador NO puede ejecutar scripts, se puede utilizar la etiqueta **noscript**:

```
<script>  
    alert("Hola!")  
</script>  
<noscript>Este navegador no soporta o no tiene habilitado los  
scripts.</noscript>
```

¿DÓNDE SE UBICAN LOS SCRIPTS?

Pueden estar incluidos directamente en el documento html o en archivos externos.

```
<html>
  <head>
    <script src="misScripts.js">
    </script>
  </head>
  <body>
    <script>
      ...código....
    </script>
  </body>
</html>
```

¿SIEMPRE JAVASCRIPT?

- Es el lenguaje de scripts por defecto.
- A partir de HTML5.2 quedó **obsoleto** especificar el lenguaje de script.
- Aunque existe el atributo **type**, no es necesario para indicar el lenguaje.

```
<script type="text/javascript" >  
...código....  
</script>
```

TYPE="MODULE"

- El atributo type puede usarse para indicar que el script es un module script.
- Un módulo es un script que puede **importarse** cuando sea necesario.
- + Info

```
<script type="module" src="mi_modulo.js"></script>>
```


JAVASCRIPT: CARACTERÍSTICAS BÁSICAS

- El estándar es el EcmaScript 262.
- JavaScript es un lenguaje de programación interpretado, multiparadigma y dinámico.
- Puede correr tanto en el cliente como en el servidor.
- **NOSOTROS lo usaremos en forma client-side**: ¿dónde está el intérprete?

FORMA DE EJECUCIÓN: ATRIBUTOS ASYNC Y DEFER



- Sacado de <https://www.josefzacek.cz/blog/whats-the-difference-between-async-vs-defer-attributes/>
- + Info

LA CONSOLA: PROBLEMAS...

```
parseInt("1234");  
parseInt("11", 2);  
parseInt("0x10");  
parseInt("hola");  
Math.sqrt(9);  
Number.MAX_VALUE;  
  
"proyecto".length;  
"proyecto".charAt(2);  
"proyecto de desarrollo".replace("desarrollo", "software");  
"proyecto".toUpperCase();  
  
var arreglo = new Array("uno", 2, "tres");  
arreglo[3] = "3333";  
  
var frutas = new Array();  
frutas["citricos"] = new Array("naranja", "pomelo");  
frutas["otros"] = new Array("manzana", "pera");  
frutas["citricos"];  
frutas.citricos;
```

SEGUIMOS PROBANDO...

- Veamos un script simple en un documento HTML.
- Más ejemplos en estos tutoriales.

FUNCIONES EN JAVASCRIPT

```
function nombreFuncion(par1, par2, par3) {  
  // código  
}
```

Algunos ejemplos

HABLAMOS DE SCRIPTS, PERO...

¿Para qué los utilizamos en nuestros desarrollos?

Si queremos cambiar el contenido dinámicamente de mi página, o hacer validaciones, entre otras muchas cosas ¿qué necesitamos?

DOM: EL MODELO DE OBJETOS DEL DOCUMENTO

DOM: EL MODELO DE OBJETOS DEL DOCUMENTO

- El modelo de objetos del documento es una **API**, que permite acceder a los contenidos de un documento HTML.
- Proporciona una **representación** estructurada, **orientada a objetos**, de los elementos individuales y el contenido del documento, con métodos para recuperar y fijar las propiedades de dichos objetos.
- Proporciona métodos para agregar y eliminar objetos al documento.
- También proporciona una **interfaz estándar** para trabajar con **eventos**.
- La especificación en: <https://dom.spec.whatwg.org/>

EL DOCUMENTO SE VE COMO UN ÁRBOL DE NODOS

- Cada nodo tiene sus propios métodos y propiedades, pero todos implementan la **interfaz Node**: un conjunto común de métodos y propiedades relacionadas con la estructura de árbol del documento.
- Por ejemplo: **insertBefore()**, **appendChild()**, **removeChild()**, entre otras.
- Por ejemplo: **firstChild**, **lastChild**, **childNodes**, **parentNode**, entre otras.

EN UN DOCUMENTO HTML

- El documento entero es un **nodo documento**.
- Cada elemento HTML es un **nodo elemento**.
- Los textos que aparecen en las páginas son **nodos de texto**.

EL DOCUMENTO

- La raíz del árbol es el **objeto document**, que implementa la interfaz Document.
- El objeto document tiene sólo un elemento hijo, dado por **document.documentElement**.
- **document.documentElement** corresponde al elemento **<html>** y tiene dos hijos: **<head>** y **<body>**

```
<html>
```

```
<head>
```

```
<title> La web </title>
```

```
</head>
```

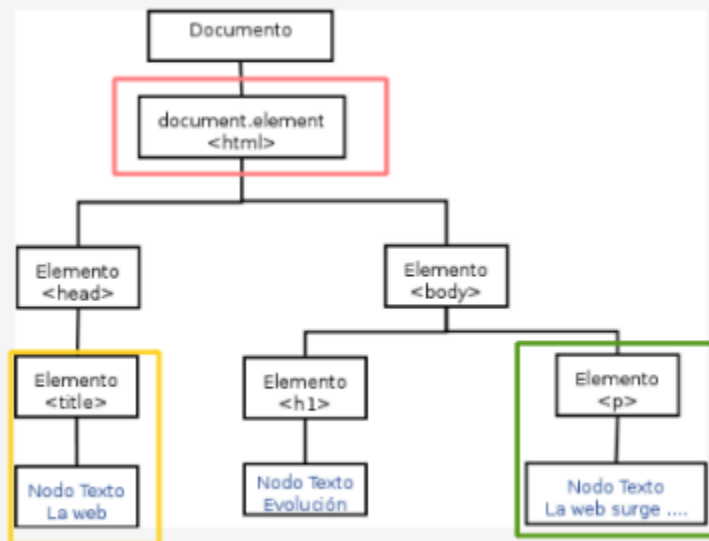
```
<body>
```

```
<h1>Evoluci&osacute;n </h1>
```

```
<p> La web surge .....</p>
```

```
</body>
```

```
</html>
```



INTERFAZ DOCUMENT

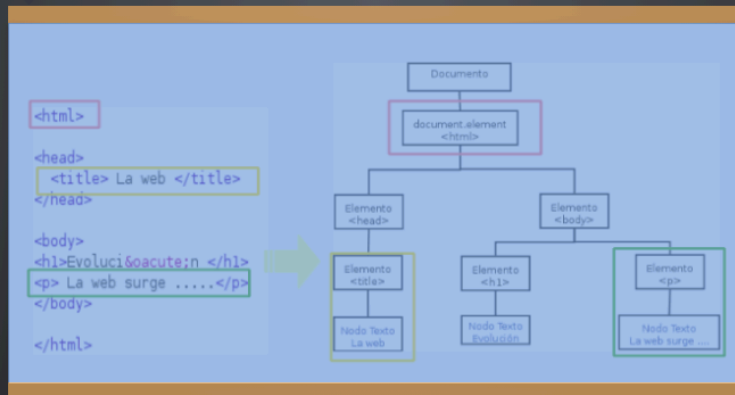
- La interfaz Document proporciona métodos para acceder y crear otros nodos en el árbol del documento tales como:
 - `getElementById()`
 - `getElementsByTagName()`
 - `getElementsByClassName()`
 - `createElement()`
 - entre otros...

DOM - ALGUNAS HERRAMIENTAS

EL DOCUMENTO

- La raíz del árbol es el **objeto document**, que implementa la **interfaz Document**.
- El objeto document tiene sólo un elemento hijo, dado por **document.documentElement**.
- document.documentElement** corresponde al elemento **<html>** y tiene dos hijos: **<head>** y **<body>**

img | 645.17 x 316.2



```
Elements
<html id="el_documento">El documento</html>
  <ul></ul>
  <p>
    
  </p>
  <section data-markdown data-markdown-parsed="true" class="future" style="top: 6px; display: block;" hidden aria-hidden="true"></section>
  <section data-markdown data-markdown-parsed="true" class="future" style="top: 99px; display: block;"></section>
  <section data-markdown data-markdown-parsed="true" class="future"></section>

#index-page body
Styles
element.style {
  transition: > -webkit-transform 0.8s ease 0s;
}
body {
  background-color: #1c1e20;
  background-image: -moz-radial-gradient(circle, circle cover, #555a5f 0%, #1c1e20 100%);
  background-image: -webkit-gradient(radial, center, center, 0px, center, 100%, color-stop(0%, #555a5f), color-stop(100%, #1c1e20));
  background-image: -ms-radial-gradient(circle, circle cover, #555a5f 0%, #1c1e20 100%);
  background-image: -o-radial-gradient(circle, circle cover, #555a5f 0%, #1c1e20 100%);
  background-image: radial-gradient(circle, circle cover, #555a5f 0%, #1c1e20 100%);
}
```

RECORRIENDO EL ÁRBOL

Miremos el siguiente código:

```
<html>
  <head>
    <title> DOM </title>
  </head>
  <body>
    <h1> Algo interesante ... </h1>
    <p>   Mmmm es interesante? </p>
  </body>
</html>
```

- ¿A qué elemento hace referencia:
document.documentElement.lastChild.firstChild.tagName?

ACCEDIENDO A LOS NODOS

- A través del atributo **id**, podemos utilizar el método:
document.getElementById()
- Para recuperar todos los elementos de un mismo tipo, se puede usar el método:
document.getElementsByTagName() o **getElementsByClassName()**.
- Veamos un **ejemplo sencillo**.

ACCEDIENDO A NODOS: QUERYSELECTOR Y QUERYSELECTORALL

```
<script>
  var primer_parrafo = document.querySelector("p.especial");
  primer_parrafo.style.color = "blue";
</script>
```

```
<script>
  var x = document.querySelectorAll("p.especial");
  for (var i = 0; i < x.length; i++)
  {
    x[i].style.color = "blue";
  }
</script>
```

MUCHAS LIBRERÍAS JS BRINDAN ATAJO

- Atajos a las funciones de DOM en JQuery:
 - `document.getElementById("p1")` vs. `$("#p1")`
 - `document.getElementsByTagName("p")` vs. `$("p")`
- JQuery usa los selectores CSS para acceder a los elementos:
 - `$("p.intro")`: todos los elementos `<p>` con `class="intro"`.
 - `$(".intro")`: todos los elementos con `class="intro"`
 - `$("p#demo")`: todos los elementos `<p>` `id="demo"`.
 - `$(this)`: el elemento actual
 - `$("ul li:odd")`: los `` impares dentro de ``

MODIFICANDO EL ÁRBOL

- La interfaz document incluye métodos que permiten modificar el árbol de nodos: **createElement**, **createTextNode**
- **Ejemplo**: quiero crear una lista en forma dinámica....

```
var lista=document.createElement("ul");  
var item=document.createElement("li");  
  
lista.appendChild(item);  
document.documentElement.lastChild.appendChild(lista);  
....
```

- Veamos este ejemplo de listas

TIPOS DE NODOS

- **Nodos elementos:** corresponden a las etiquetas del documento. Pueden tener nodos hijos, que pueden ser otros elementos o nodos de texto.
- **Nodos de texto:** representan contenido, o simplemente caracteres. Tienen un nodo padre y, posiblemente, nodos del mismo nivel, pero no pueden tener nodos hijos.
- En realidad hay más...

NODOS DE TEXTO

- Los nodos de texto no tienen un atributo **id**.
- No se pueden acceder mediante los métodos **getElementById()** o **getElementsByName()**.
- **Se acceden a través de su nodo padre.**
- Ejemplo:

```
<p id='p1'> Texto inicial ....</p>  
<script>  
  document.getElementById('p1').firstChild.nodeValue='Otro';  
</script>
```

- **Completamos** el ejemplo anterior.
- nodeValue vs innerHTML.

DOM Y EVENTOS

DOM TAMBIÉN CONTEMPLA...

... un sistema de eventos genérico que permita registrar **manejadores** de eventos, describir el **flujo de eventos** a través de la estructura del árbol y proveer **información contextual** sobre cada evento.

- También define un subconjunto común de los sistemas de eventos actuales.

MODELO DE EVENTOS

- ¿Qué es un evento? ¿Cuándo se produce?
- La especificación de DOM:
 - Define y explica la propagación y registro de eventos.
 - Define la **interfaz Event**.
 - Define cómo se interpreta el flujo de eventos una vez producido.

FLUJO DE EVENTOS

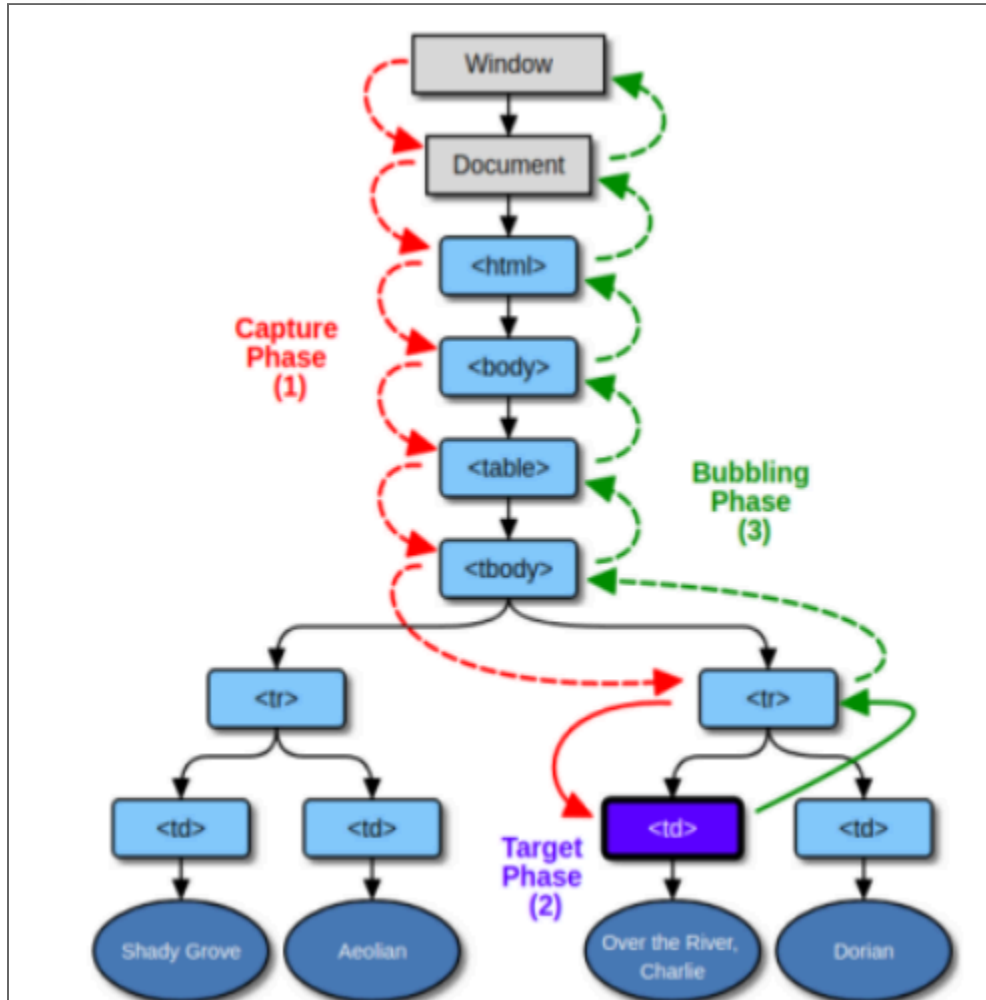


Imagen obtenida de w3.org

MANEJADORES DE EVENTOS

```
function manejador(evento) {  
    //  
    // "evento" se crea implícitamente y contiene la  
    // info sobre el evento producido.  
    //  
}  
var elem=document.getElementById('p1');  
elem.onmouseover = manejador;
```

manejador es un objeto function.

ESCUCHADORES DE EVENTOS

- Los objetos DOM también pueden ser registrados como **escuchadores de eventos**.
- Esta característica puede ser utilizada para asignar múltiples manejadores para un evento dado.
- Los métodos básicos son: **addEventListener** y **removeEventListener**

ESCUCHADORES DE EVENTOS: EJEMPLO

```
var elem = document.getElementById('p1');  
elem.addEventListener("mouseover", f1, true);  
elem.addEventListener("mouseout", f2, true);  
.....  
  
elem.removeEventListener("mouseover", f1, true);  
  
elem.addEventListener("mouseover", f1, true);  
elem.addEventListener("mouseover", f1, false);
```

UN POCO MÁS DE JAVASCRIPT

EXCEPCIONES EN JAVASCRIPT

- Se pueden lanzar excepciones usando la sentencia **throw**.
- Se manejan con la sentencia **try...catch**.
- Un ejemplo de la w3schools
- **finally** opcional.
- Más info [developer.mozilla](#)

OBJETOS Y JAVASCRIPT

```
var mi_cancion = {  
  "titulo": "Ruta 66",  
  "interprete": "Pappo",  
}
```

```
function Musica(titulo, interprete){  
  this.titulo=titulo;  
  this.interprete=interprete;  
}  
var mi_musica= new Array();  
mi_musica[0]= new Musica(  
  "Desconfio",  
  "Pappo");  
mi_musica[1]= new Musica(  
  "Sussy Cadillac",  
  "Riff");  
mi_musica[2]= new Musica(  
  "Llegara la paz",  
  "Pappo's Blues");
```

- ¿Qué representa la función Musica?

• ¿this?

AGREGANDO MÉTODOS

```
function Musica(titulo, interprete){
    this.titulo=titulo;
    this.interprete=interprete;
    this.es_solo_Pappo = function(){return (this.interprete
== "Pappo")};
}
var mi_musica= new Array();
mi_musica[0]= new Musica(
    "Desconfio",
    "Pappo");
mi_musica[1]= new Musica(
    "Sussy Cadillac",
    "Riff");
mi_musica[2]= new Musica(
    "Llegara la paz",
    "Pappo's Blues");

for (var i = 0; i<mi_musica.length; i++){
    objeto = mi_musica[i];
    if (objeto.es_solo_Pappo()){
        console.log(objeto.titulo)}
}
```

- Probamos el ejemplo.

OBJETOS Y JAVASCRIPT

Ahora con class

```
class Musica{  
  constructor (titulo, interprete){  
    this.titulo=titulo;  
    this.interprete=interprete;  
  }  
  es_solo_Pappo () {  
    return this.interprete == "Pappo";  
  }  
}
```

- ¿Qué diferencias hay con la función Musica?
- ¡Las clases NO son objetos!
- Más info en este [artículo sobre modelo de objetos](#).
- Y en la [definición del estándar](#)

GETTERS Y SETTERS

```
class Musica{
  constructor (titulo, interprete){
    this.titulo = titulo;
    this.interprete = interprete;
    this.language = ""
  }
  get es_solo_Pappo () {
    return this.interprete == "Pappo";
  }
  get resumen () {
    return this.titulo + ":" + this.interprete + "
    (" + this.language + ")";
  }
  set idioma (idioma) {
    this.language = idioma;
  }
}

var objeto = new Musica("El Hombre Suburbano", "Pappo's Blues");
console.log(objeto.resumen);
objeto.idioma = "Español";
console.log(objeto.resumen);
```

+ Info en este ejemplo de w3school

EL PROTOTIPO

Recordemos que *JavaScript es un lenguaje basado en prototipos.*

```
class Musica{
  constructor (titulo, interprete){
    this.titulo = titulo;
    this.interprete = interprete;
  }
  get resumen () {
    return this.titulo + ":" + this.interprete + "
    (" + this.album + ")";
  }
}

var objeto = new Musica("Ruta 66", "Pappo");
var objeto1 = new Musica("Mi vieja", "Pappo");
console.log(objeto.resumen);
Musica.prototype.album = "Blues local"
console.log(objeto.resumen);
```

- Probemos lo mismo con objeto1. ¿Qué efecto tiene modificar la propiedad **prototype**?

HERENCIA

- Todos los objetos JavaScript heredan propiedades y métodos de un prototipo:
- Los objetos Array heredan de Array.prototype.
- Los objetos Musica heredan de Musica.prototype.
- **Object.prototype** está al tope de la cadena de herencia de prototipos.

```
console.log(Musica.prototype);
```

CREANDO SUBCLASES

```
class Jugador {
  constructor(nombre, juego){
    this.nom = nombre;
    this.juego = juego;
  }
  get nombre() {
    return this.nom;
  }
}
class JugadorDeFIFA extends Jugador {
  constructor(nombre, plataforma){
    super(nombre, 'FIFA');
    this.dispositivo = plataforma;
  }
  get plataforma(){
    return this.dispositivo;
  }
}
var nico = new JugadorDeFIFA('Nico Villalba', 'PS4');
console.log(nico.plataforma);
console.log(nico.nombre);
```

- ¿super()?

- ¿JugadorDeFIFA.prototype?

Y SURGIÓ JSON ...

- JSON: JavaScript Object Notation.
- Surge como una forma de definir objetos en JavaScript.

```
var mi_cancion = {  
  "título": "Lily Malone",  
  "interprete": "Riff",  
}
```

- Es un formato ligero para el envío y recepción de datos.

JSON

- JSON se basa en dos estructuras:
 - Objetos: una colección de pares de nombre/valor
 - Arreglos: una lista ordenada de objetos

```
pappo1 = {  
  "titulo": "Blues para mi guitarra",  
  "interprete": "Pappo",  
}  
pappo2 = {  
  "titulo": "El hombre suburbano",  
  "interprete": "Pappo",  
}  
coleccion= [pappo1, pappo2];
```

```
var mi_musica = [  
  {titulo: "Desconfio",  
   interprete: "Pappo"},  
  {titulo: "Sussy Cadillac",  
   interprete: "Riff"},  
  {titulo: "Llegará la paz",  
   interprete: "Pappo's Blues"}  
]
```

1;

MUY USADO

- Veamos este ejemplo:
https://developers.mercadolibre.com.ar/es_ar/categoriza-productos
- ¿Conocen la central meteorológica de la facultad? Miremos estos datos
- Hay alternativas como XML y YAML.

LIBRERÍAS/Frameworks JAVASCRIPT

- En javascripting.com hay una lista de más de 1000 registradas.
- No todas con el mismo objetivo.
- Para desarrollo en los últimos años...

METEOR



aurelia



BACKBONE.JS

¿JQUERY EN LA ACTUALIDAD?

Para desarrollos sencillos

- Aún actualmente, muy usada:
https://w3techs.com/technologies/overview/javascript_library/all
- Atajos a las funciones de DOM:
 - `document.getElementById("p1")` vs. `$("#p1")`
 - `document.getElementsByTagName("p")` vs. `$("p")`
- JQuery usa los selectores CSS para acceder a los elementos:
 - `$("p.intro")`: todos los elementos `<p>` con `class="intro"`.
 - `$(".intro")`: todos los elementos con `class="intro"`
 - `$("p#demo")`: todos los elementos `<p>` `id="demo"`.
 - `$(this)`: el elemento actual
 - `$("ul li:odd")`: los `` impares dentro de ``

SEGUIMOS LA PRÓXIMA MEJOR...

Speaker notes