



***¡LES DAMOS LA
BIENVENIDA!***

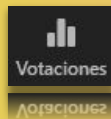
¿Están listos?

PRESENTACIÓN DE ESTUDIANTES



Por encuestas de Zoom:

1. País
2. Conocimientos previos en la disciplina
3. ¿Por qué elegiste el curso?



Se responde en encuestas separadas creadas por el docente.



¿DUDAS DEL ON-BOARDING?

MIRALO ACA

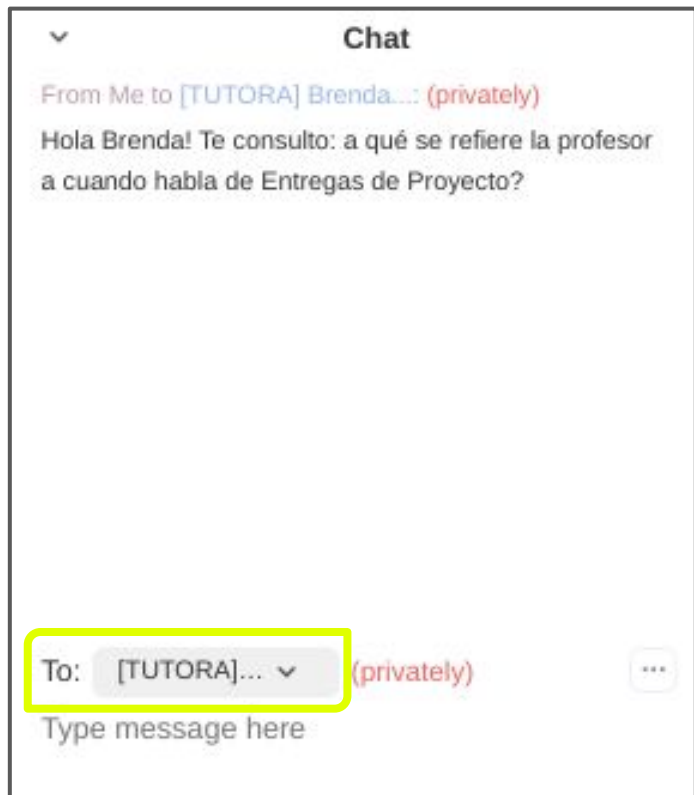
INTERACCIÓN EN CLASE

Mientras el docente explica...

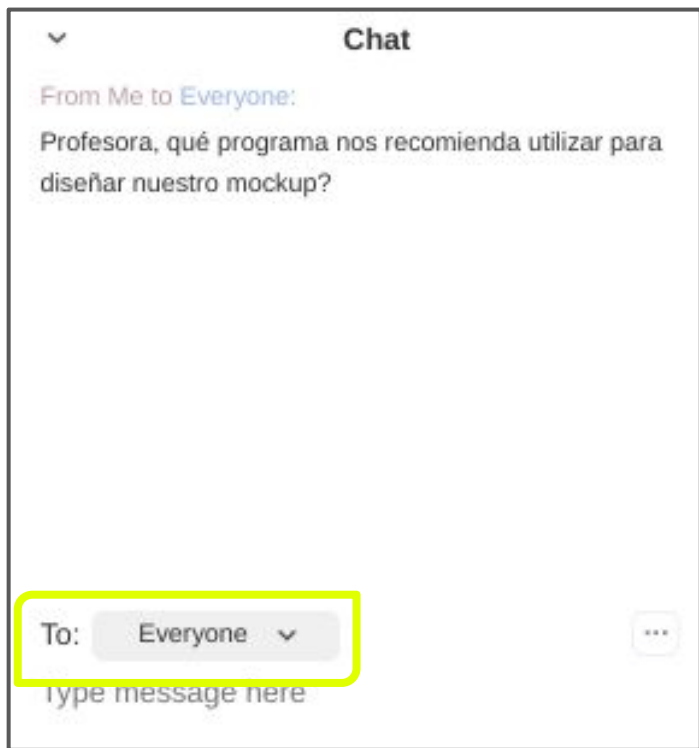
Para mantener una **comunicación clara y fluida** a lo largo de la clase, establecemos dos reglas:

1

Si tenés **dudas mientras el docente explica**, deberás consultarle directamente **por privado a tu tutor** por el chat de Zoom.



Cuando el docente abre un espacio de consultas



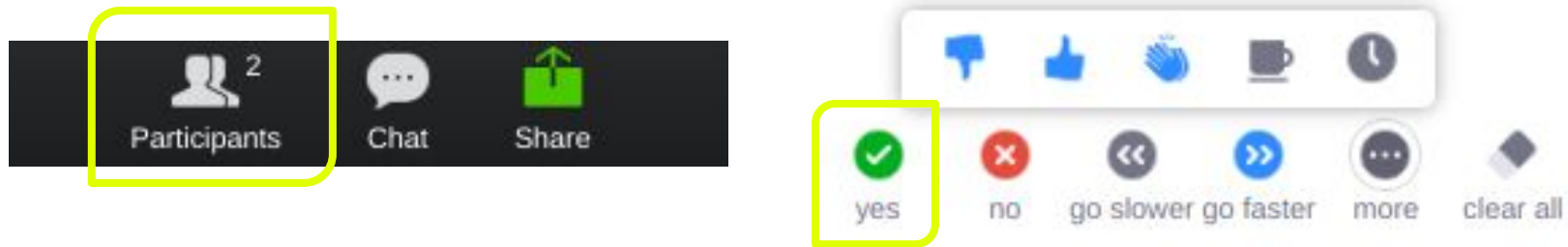
2

Entre contenido y contenido, **el docente abrirá breves espacios de consulta.**

Allí podrás escribir en el chat tu pregunta para que el docente pueda leerla.

¡Ojo! No te olvides de seleccionar **“todos”** para que todos puedan leerte (y no solo tu tutor). ¡Tu duda puede ayudar a otros!

Para evitar saturar el chat de mensajes, **utilizá los signos que figuran en el apartado de “participantes”, dentro de Zoom**.**

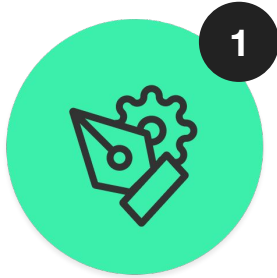


Por ejemplo: si el docente pregunta si escuchan su audio correctamente, deberán seleccionar la opción “SI” o “NO”.

**** Para quitar el signo, presioná el mismo botón nuevamente o la opción “clear all”.**

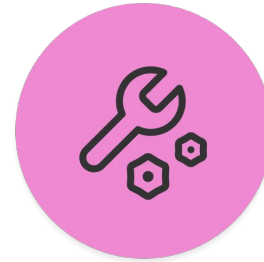
DESAFÍOS

Son actividades o ejercicios que se realizan en clase para que la misma pueda enfocarse en la práctica. Existen dos tipos de desafíos:



Desafíos entregables

Relacionados completamente con el Proyecto Final. Deben ser subidos obligatoriamente a la plataforma para que el docente pueda corregirlos.



Desafíos genéricos

Ayudan a poner en práctica los conceptos y la teoría vista en clase. No deben ser subidos a la plataforma.

¡IMPORTANTE!

Los desafíos tienen fecha de entrega 7 días después de finalizada la clase. Te sugerimos llevar las entregas al día.

LUNES 16/03 20:30HS

10. Estrategia de contenido para Twitter y LinkedIn



● HOY 20:30



Tenes tiempo hasta el
23/03/2020

DESAFÍO - EXPIRA EL 23/03/2020

Crear publicaciones para Twitter

↑ ENTREGAR

PROYECTO FINAL

El **proyecto final** se debe subir a la plataforma la ante-última o última clase del curso. *En caso de no hacerlo tendrás 20 días a partir de la finalización del curso para cargarlo en la plataforma*, pasados esos días el botón de entrega se inhabilitará.

Es la sumatoria de los **desafíos** que se realizan clase a clase. Se va creando a medida que el estudiante sube los desafíos entregables a nuestra plataforma.

El objetivo es que cada estudiante pueda utilizar su Proyecto Final como parte de su portfolio personal.

¿CUÁL ES NUESTRO PROYECTO FINAL?

CODER HOUSE

CREA TU PROPIA TIENDA ECOMMERCE

Desarrollarás el front end de una tienda online con carrito de compras, utilizando los componentes React y Firebase como servidor en la nube. Crearás así una experiencia de usuario amigable con actualizaciones visuales instantáneas y código escalable.



Clase 01. React JS

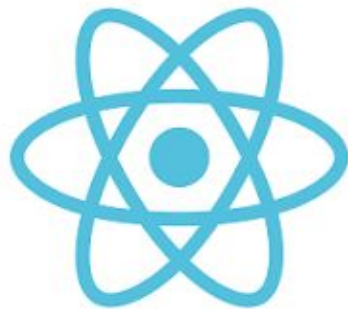
NIVELACIÓN



OBJETIVOS DE LA CLASE

- Repasar HTML, CSS y fundamentos JavaScript.
- Conocer la sintaxis de JavaScript y ECMAScript 6
 - Comprender el uso de la variable 'this'
 - Realizar una introducción a React JS

NIVELACIÓN HTML, CSS Y JS



React JS es una biblioteca para **desarrollo web**, por lo cual debemos contar con conocimientos mínimos sobre los lenguajes que el navegador web interpreta.

HTML

CODER HOUSE

HTML es un **lenguaje de etiquetas**, el cual dará la estructura para nuestras páginas web.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Curso de React</title>
</head>
<body>
  <p>Esta es la primera clase</p>
</body>
</html>
```

HTML 5

HTML 5 es una nueva versión de diversas especificaciones, entre las que se encuentran:

- HTML 4
- XHTML 1
- CSS Nivel 2
- DOM Nivel 2 (DOM = Document Object Model)



Cuáles son las novedades de HTML 5

HTML 5 incluye novedades significativas en diversos ámbitos. Este nuevo estándar supone mejoras en áreas que hasta ahora quedaban fuera del lenguaje y para las que se necesitaba utilizar otras tecnologías:

- Estructura del `<body>`
- Etiquetas para contenido específico
- Canvas
- Bases de datos locales
- Web Workers
- Aplicaciones web Offline
- Geolocalización

Un poco más de HTML

Ahora que ya entendemos qué es HTML y qué es HTML 5, comencemos a ver un poco más de código.

Este es un ejemplo de código en HTML:

```
<div class="site-logo col-6">
  <a href="index.html">
    
  </a>
</div>
<nav class="site-navigation">
  <ul class="site-menu js-clone-nav d-none d-lg-block">
    <li><a href="#acerca" class="nav-link">ACERCA DEL ARTISTA</a></li>
    <li><a href="#music">MÚSICA</a></li>
    <li><a href="#video">COMPOSICIONES AUDIOVISUALES <span class="ml-1 mr-1">/>
    <li><a href="#prensa">PRENSA</a></li>
  </ul>
</nav>
```

DOCTYPE

DOCTYPE no es una etiqueta, sino una instrucción para **indicar al navegador qué versión de HTML vamos a utilizar.**

El DOCTYPE mostrado en nuestro ejemplo anterior es del estándar HTML 5.

```
<!DOCTYPE html>
```

Metadatos del documento

- **<title>**: Define el título del documento, el cual se muestra en la barra de título del navegador o en las pestañas de página.
- **<base>**: Define la URL base para las URLs relativas en la página.
- **<link>**: Utilizada para enlazar JavaScript y CSS externos.
- **<meta>**: Con esta etiqueta definimos la codificación que tendrá nuestro archivo, los mismos pueden ser:
 - UTF-8
 - ANSI

ANSI es el formato estándar de codificación de archivos utilizados en el Bloc de notas.

Scripting

- **<script>**: Define tanto un script interno como un enlace hacia un script externo. El lenguaje de programación es JavaScript
- **<noscript>**: Define un contenido alternativo a mostrar cuando el navegador no soporta scripting.

DOM

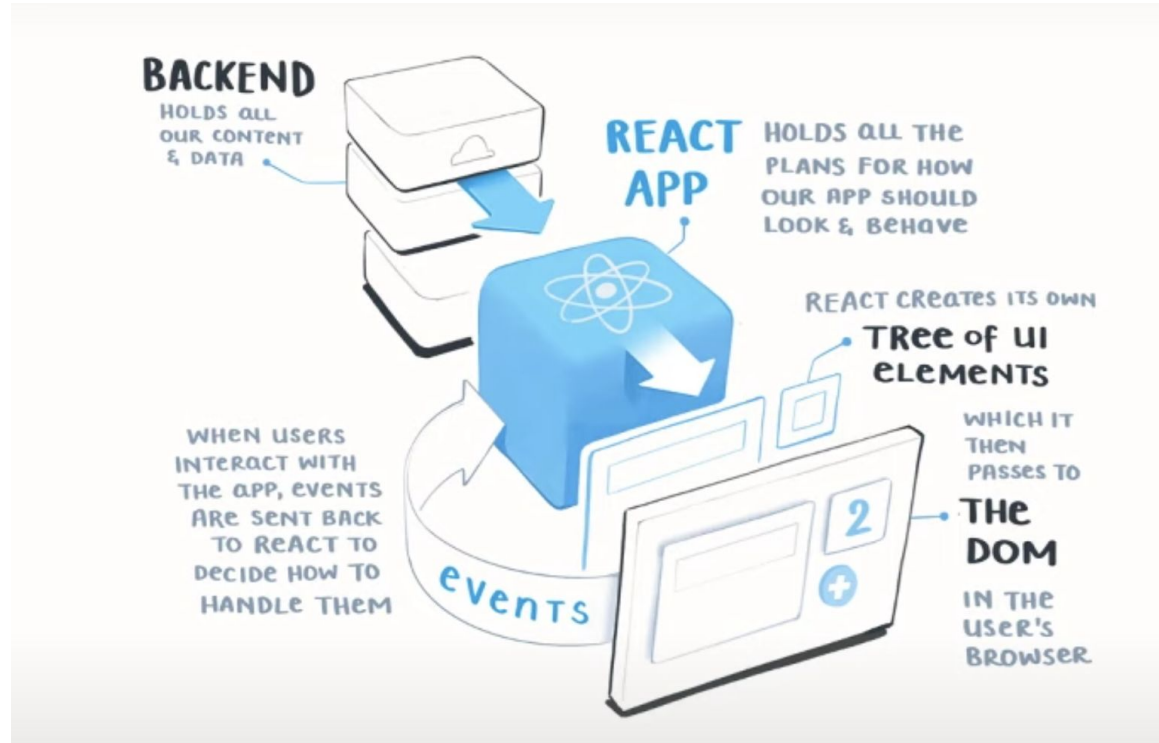
CODER HOUSE

DOM

DOM (Document Object Model o modelo de objetos del navegador) nos sirve para **acceder** a cualquiera de los **componentes** que hay dentro de una página.

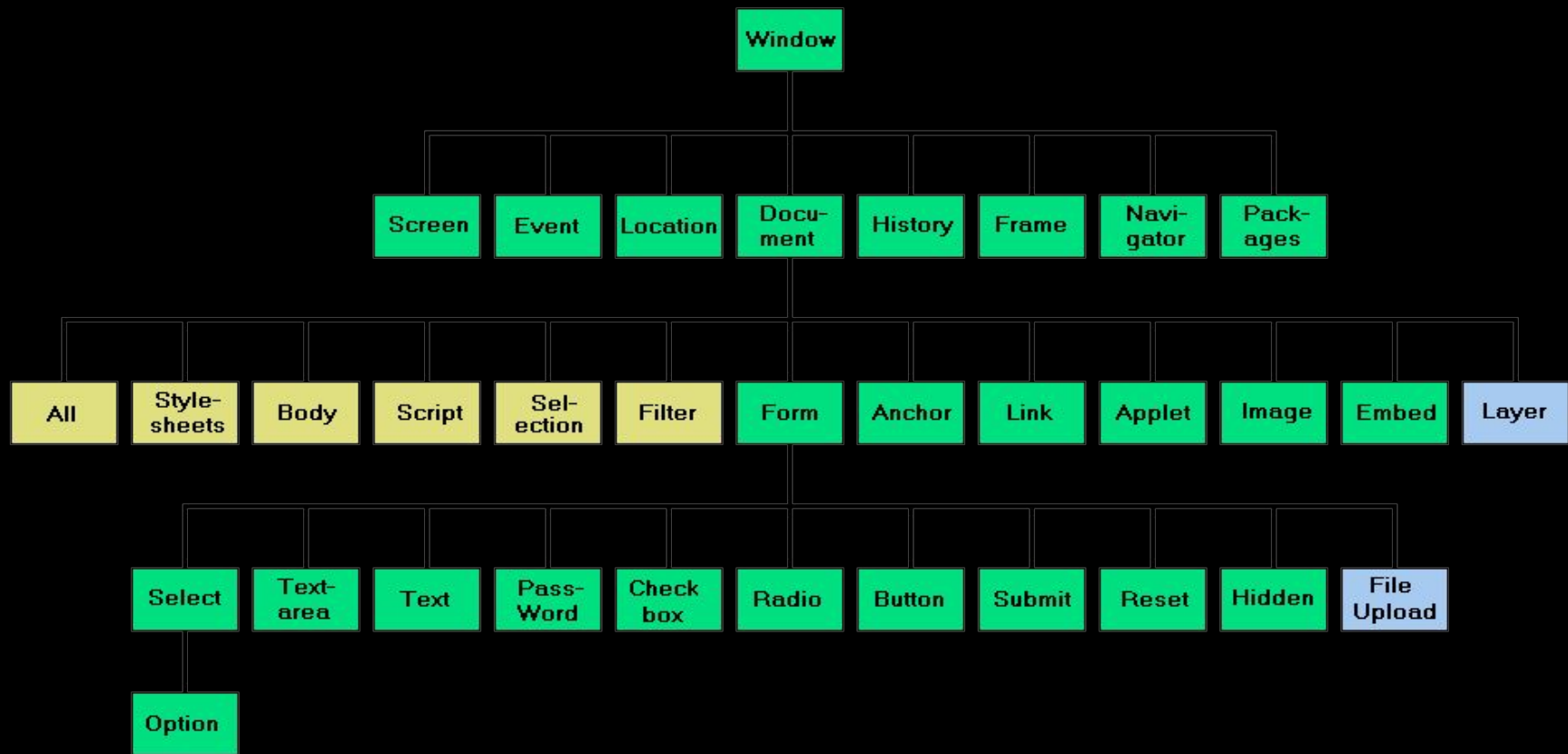
Por medio del DOM podremos **controlar al navegador** en general y a los distintos elementos que se encuentran en la página.

DOM como metáfora



Ilustradora: [Maggie Appleton](#) @ Woman of React 2020

CODER HOUSE




Jerarquía del DOM

Un mundo de API's

Siempre es recomendable **avanzar con criterio** y ver qué funcionalidades usamos del **Navegador**, ya que hay centenas de API's, pero:

1. No todas están listas para **producción**
2. No todas tienen soporte en todos los navegadores

Especificación	Estado	Comentario
Geolocation API	 REC Recommendation	Especificación inicial

Specification	Status	Comment
Web Animations The definition of 'AnimationEffect' in that specification.	 WD Working Draft	Editor's draft.

<https://developer.mozilla.org/es/docs/Web/API>

Un mundo de API's

Todo depende del **contexto** del trabajo y el **target** de usuarios / plataformas.

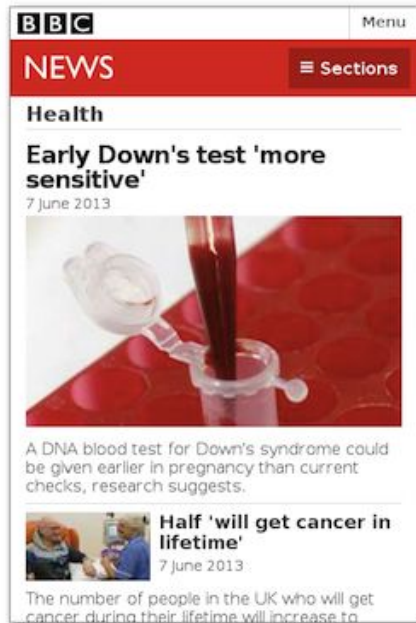
Puedo emocionarme para luego enterarme que algunos usuarios no pueden utilizar la experiencia :(

Desktop						Mobile					
Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet
75	79	63	No	62	13.1	75	75	63	54	13.4	11.0
75	79	63	No	62	13.1	75	75	63	54	13.4	11.0
75	79	63	No	62	13.1	75	75	63	54	13.4	11.0
75	79	63	No	62	13.1	75	75	63	54	13.4	11.0

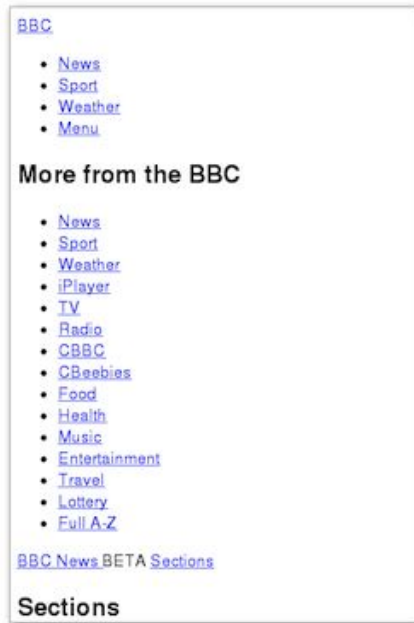
<https://developer.mozilla.org/es/docs/Web/API>

CSS

CODER HOUSE



Con CSS



Sin CSS

CSS (cascading style sheets - hojas de estilo en cascada) es un lenguaje de diseño gráfico con el cual podremos **dar estilos** (diseño, colores, márgenes) a nuestras webs desarrolladas con HTML. Veremos la sintaxis básica de CSS

Incluir una hoja de estilo en el HTML

Para incluir una hoja de estilo en nuestro documento HTML podemos utilizar el elemento `<link>`.

Dentro del archivo CSS externo incluiremos las sentencias correspondientes, como lo hacemos con un archivo css en línea.
Por ejemplo:

```
<!-- MAIN CSS -->  
<link rel="stylesheet" href="css/style.css">
```

Clases CSS

Podemos seleccionar uno o más elementos html a través de su **clase** o su **atributo “id”** en CSS. O todos los elementos de un **tipo**

Modos básicos de “**seleccionar/capturar**” elementos por su:

Tipo de elemento (div, ul, li, nav, input)	div { background: lightgray; }
ID (usando # adelante)	#nombre { background: coral; }
Clase (usando punto inicial)	.apellido

Classes CSS

Elemento

```
<div>@gaearon</div>
```

Id

```
<div id="nombre">Dan</div>
```

Clase

```
<div class="apellido">Abramov</div>
```

Clases CSS

Error clásico:

Los # y los puntos “.” de los selectores se usan **en la hoja de estilos**, **NO** en el **elemento**

```
X  
<div id="#nombre">Dan</div>  
X  
<div class=".apellido">Abramov</div>
```

BREAK



CODER HOUSE

JAVASCRIPT

CODER HOUSE

Javascript es el **lenguaje de programación web** por excelencia.

Decimos que se trata de un lenguaje de programación **interpretado**.

Su uso más conocido es del lado del cliente (**client-side**), corriendo en el navegador web, permite mejoras en la interfaz de usuario.

React está desarrollado en JS, por eso lo llamamos **reactjs**.



¡Pregunta rápida!

“En una propuesta laboral decía que necesitaba saber Vanilla JS, ¿Dónde lo puedo aprender?”

¡Tal como el helado más **común**, el saborizante más **conocido**, o el clásico de los clásicos!

¡JS sin aditivos ni conservantes!

(O mejor dicho sin librerías externas)



Incluir el archivo JS en nuestro HTML

```
43  
44 <script type="text/javascript" src="prueba.js"></script>  
45
```

Esto debe ser incluido en el head del html. También podemos introducir nuestro código javascript dentro de la etiqueta

`<script></script>`. Por ejemplo:

```
<script>  
  function mi_funcion() {  
    |   document.getElementById("mi_funcion").innerHTML = "Ejemplo de mi funcion"  
  }  
</script>
```

Tipos de ejecución: Ejecución directa

Es el método de ejecución de scripts más básico. En este caso se incluyen las **instrucciones dentro de la etiqueta <script>**, y cuando el navegador lee la página y encuentra un script va interpretando las líneas de código y las va **ejecutando una después de otra.**

Tipos de ejecución: Respuesta a un evento

Los eventos son **acciones que realiza el usuario**. Los programas como Javascript están preparados para atrapar determinadas acciones realizadas, en este caso sobre la página, y **realizar acciones como respuesta**.
Por ejemplo la pulsación de un botón, el movimiento del ratón o la selección de texto de la página.

Variables

La manera estándar de JS de **almacenar valores** en memoria es a través del uso de variables

```
var nombre = 'dan';  
  
let apellido = 'abramov';
```

Definir una constante

Desde ES6 (ECMA 2015) podemos definir una constante con la palabra reservada `const`

```
38  const PI = 3.141593;  
39  alert(PI > 3.0);  
40
```

Las **const** no pueden cambiar la referencia con la que fueron inicializadas.



¡Recuerden!

Puede ser poco intuitivo, pero...
Que una constante mantenga para siempre
la misma referencia no significa que esa
referencia tenga el mismo contenido, y de
hecho

¡Puedo cambiarlo!

DOM

DOM (Document Object Model o modelo de objetos del documento) nos sirve para **acceder** a cualquiera de los **componentes** que hay dentro de una página.

Por medio del DOM podremos **controlar al navegador** en general y a los distintos elementos que se encuentran en la página.

Let

Podemos utilizar `let` en lugar de `var` a la hora de declarar una variable, cuando queremos que esta solo sea accedida de manera local en determinado ámbito. Por ejemplo:

```
55  
56 //ES6  
57 let nombre1 = 'Javascript';  
58 let nombre2 = 'awesome';  
59 console.log('Solo quiero decir que $(nombre1) es $(nombre2)');  
60 // Solo quiero decir que Javascript es awesome  
61  
62
```


Función Arrow

En ambos ejemplos se sustituye el uso de la palabra reservada **function** dando simplicidad al código.

(parametro1,parametro2,...,parámetro n) => {Definición de la función}

```
//ES6
var miFuncion = (num) => num + num;

//ES6
var data = [{...}, {...}, {...}];
data.forEach(elem => {
  console.log(elem);
})
```

JAVASCRIPT: La variable 'this'

This

La variable **this** está mucho más relacionada con el **entorno y/o contexto de ejecución** que con el concepto de esta misma variable. This representa al objeto en sí.

This - Ejemplo

Por ejemplo: en el siguiente código, si no hacemos `var that = this` dentro de la función `document.addEventListener`, `this` haría referencia a la función que pasamos por Callback y no podríamos llamar a `foo()`

```
//ES3
var obj = {
  foo : function() {...},
  bar : function() {
    var that = this;
    document.addEventListener('click', function(e) {
      that.foo();
    });
  }
}
```

This - ECMAScript5

```
//ES5
var obj = {
  foo : function() {...},
  bar : function () {
    document.addEventListener('click', function(e) {
      this.foo();
    }).bind(this);
  }
}
```

Con ECMAScript5 la cosa cambió un poco, y gracias al método `bind` podíamos indicarle que `this` hace referencia a un contexto y no a otro.

This - ES6

```
//ES6
var obj = {
  foo : function() {...},
  bar : function () {
    document.addEventListener('click', (e) => this.foo());
  }
}
```

Con ES6 y la función `Arrow =>`
la cosa es todavía más visual y
sencilla.

Clases

JavaScript también tiene clases. Estas son muy parecidas a las funciones constructoras de objetos que realizamos en el estándar anterior, pero ahora bajo el paradigma de clases, con todo lo que eso conlleva (ejemplo: herencia)

Sugar-syntax

```
class Animal {  
  constructor(sonido = 'ola-k-ac') {  
    this.sonidoPrimitivo = sonido;  
  }  
  
  emitirSonido() {  
    console.log(this.sonidoPrimitivo);  
  }  
}
```

```
class Persona extends Animal {  
  constructor(altura) {  
    this.altura = 1.20  
    super('Hola! Soy una persona!');  
  }  
  
  hablar() {  
    return this.emitirSonido();  
  }  
  
  getAltura() {  
    return this.altura;  
  }  
  
  correr() {  
    console.log('Hey! Vamo a correr');  
  }  
}
```

VAMOS AL CÓDIGO



CODER HOUSE

Template Strings

Antes de **ES6** creábamos mensajes compuestos de la siguiente manera

```
var nombre = 'dan';  
  
let apellido = 'abramov';  
  
const cuenta = '@gaeareon'  
  
console.log('Mi nombre es ' + nombre + ' ' +  
apellido + '\n encuéntrenme en ' + cuenta)
```

console was cleared

Mi nombre es dan abramov
encuéntrenme en @gaeareon

>

Problema: ¿Qué problemas detectan?

Agregar espacios manuales entre concatenaciones '+', dificultad para modificar texto, multiline explícito \n, poca legibilidad :(

Template Strings

Con **ES6** podemos interpolar **strings** de una forma más sencilla que como estábamos haciendo hasta ahora. Miremos el ejemplo:

```
var nombre = 'dan';  
  
let apellido = 'abramov';  
  
const cuenta = '@gaearon'  
  
console.log(`Mi nombre es ${nombre} ${apellido}  
encuéntrenme en ${cuenta}`);
```

 Preview (local)  ☒ Clear

Console was cleared

Mi nombre es dan abramov
encuéntrenme en @gaearon

>

Valores por defecto / Antes y Hoy

```
function saludaLoA(nombre) {  
  console.log(nombre || 'coder');  
}
```

Antes teníamos que comprobar si la variable ya tenía un valor para dar una respuesta por defecto.

Hoy podemos usar otra sintaxis aunque tenga algunas diferencias de comportamiento

Valores por defecto: ¡Cuidado!

```
const saludaloA = (nombre = 'coder') => {  
  console.log(nombre);  
};  
  
saludaloA('dan');  
saludaloA(null);  
saludaloA(undefined);  
saludaloA();
```



Preview (local)



☒ Clear console on reload

Console was cleared

dan

null

coder

coder



En muchas convenciones **null** es considerado como algo explícito de un valor que **es la nada**. Mientras que al **undefined** se lo considera como un valor que aún no se puede inferir que es, cuando no pasamos un parámetro será **undefined** (indefinido)

Dicho esto, si le paso **null** a **function**, ¡el valor por defecto no se aplicará!

Módulos

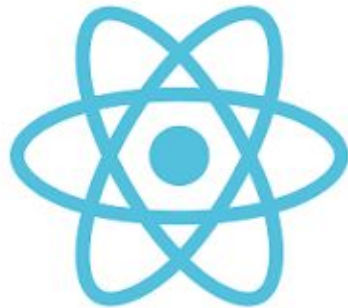
Ahora JavaScript se empieza a parecer a lenguajes como Python o Ruby. **Llamamos a las funciones desde los propios Scripts**, sin tener que importarlos en el HTML si usamos JavaScript en el navegador.

```
83
84  module 'person' {
85      export function hello(nombre) {
86          return nombre;
87      }
88  }
89
```

Y para importar en otro fichero:

```
91 import { hello } from 'person';
92 var app = {
93     foo: function() {
94         hello('Niko');
95     }
96 }
97 export app;
98
```

REACT JS: LOS INICIOS



React JS fue creada por Jordan Walke, un ingeniero de software en Facebook, inspirado por los problemas que tenía la compañía con el mantenimiento del código de los anuncios dentro de su plataforma. React intenta ayudar a los desarrolladores a **construir aplicaciones que usan datos que cambian todo el tiempo**. Su objetivo es ser **sencilla, declarativa y fácil de combinar**.

Sitios basados en React JS



NETFLIX



y más...

React utiliza una sintaxis parecida a HTML, llamada JSX que hace el código más legible, y escribirlo es una experiencia similar a HTML.

```
react.js
1  import React from 'react'; //Se importa React
2
3  class Componente extends React.Component{ //Declaración de un componente con el nombre de "Componente".
4    render(){
5      //Dentro de return se escribe en una sintaxis muy parecida a HTML
6      return(
7        <div>
8          <h1>Hola mundo</h1>
9        </div>
10     );
11   }
12 }
13
14 ReactDOM.render(<Componente />, document.getElementById('MiAppDeReact'))
15
```

Sintaxis básica de React JS

Este primer método no es la forma más popular de configurar React pero será familiar y fácil de entender.

Comencemos haciendo un archivo básico llamado **index.html**.
Vamos a cargar tres CDN en head-React, React DOM y Babel.

```
<script src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js">
</script>
<script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
```

También vamos a hacer un **div** con una identificación llamada **root**, y finalmente crearemos una etiqueta **script** donde vivirá el código de React.

Ahora, escribamos nuestro primer bloque de código de React. Vamos a usar las clases de ES6 para crear un componente React llamado **App**.

```
class App extends React.Component {  
  //...  
}
```

Ahora agregaremos el método **render()**, el único método requerido en un componente de clase, que se usa para representar nodos DOM.

```
class App extends React.Component {  
  render() {  
    return (  
      //...  
    );  
  }  
}
```

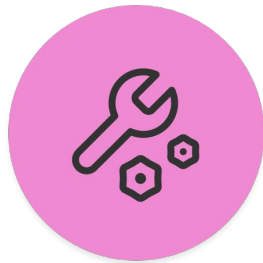
Dentro del return, vamos a poner lo que parece un simple elemento HTML

```
class App extends React.Component {  
  render() {  
    return <h1>Hello world!</h1>  
  }  
}
```

Finalmente, vamos a usar el método render() React DOM para representar la clase que creamos en el div en nuestro HTML.

```
ReactDOM.render(<App />, document.getElementById('root'))
```

¡Ahora mirá tu index.html en el navegador para ver la etiqueta H1 que creamos renderizada en el DOM!



FORMULARIO

Con lo visto hasta ahora, crear un formulario de contacto utilizando HTML5, CSS, y añadir JS para simular su funcionamiento.

Tiempo: 25 minutos

CODER HOUSE

CONCLUSIONES



RECURSOS DE LA CLASE

- <https://es.wikipedia.org/wiki/HTML>
- <http://www.w3schools.com/html/>
- <http://www.w3schools.com/css>
- https://developer.mozilla.org/es/docs/Tools/Page_Inspector
- <https://developer.mozilla.org/es/docs/Web/API>

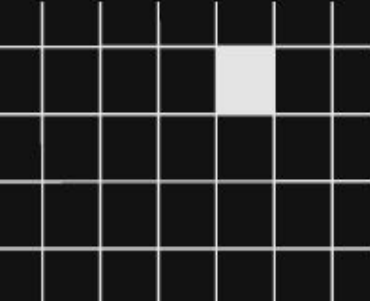
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Nivelación HTML, CSS, y JS
 - Introducción a ReactJS
- 



OPINÁ Y VALORÁ ESTA CLASE