



**¡Les damos la  
bienvenida!**

¿Comenzamos?

Esta clase va a ser

- grabada



# Encuesta After Class

Por encuestas de Zoom

¡Coordinamos nuestro PRIMER After Class!

Clase 03. REACT JS

# JSX y WEBPACK

# Temario

02

## Instalación y configuración del entorno

- ✓ Funcionamiento de React Js
- ✓ ¿Qué es virtual DOM?
- ✓ ¿Qué es NODE?
- ✓ NODE JS ¿Qué es NPM?
- ✓ Crear una aplicación utilizando el CLI

03

## JSX y Webpack

- ✓ [Sugar Syntax](#)
- ✓ [Polyfills y la retrocompatibilidad](#)
- ✓ [Bundling con Webpack](#)
- ✓ [Transpiling](#)
- ✓ [Jsx](#)

04

## Componentes I

- ✓ Componentes I: Introducción
- ✓ Composición de componentes

# Objetivos de la clase

- **Entender** las aristas del sugar syntax como proceso evolutivo de los lenguajes
- **Expandir** nuestra sintaxis avanzada de JavaScript.
- **Conocer** el rol de webpack y babel en el bundling/retrocompatibilidad.
- **Desarrollar** código en JSX.

## CLASE N°#2

# Glosario

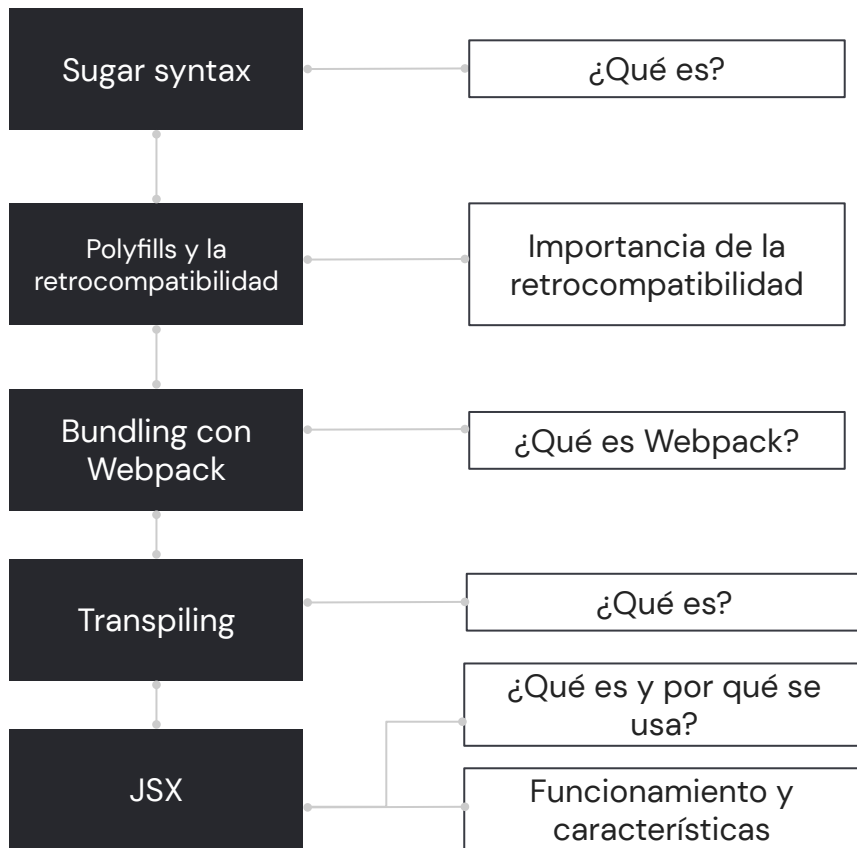
**Virtual DOM:** Es un patrón de comportamiento y React lo implementa con una tecnología llamada "Fiber". En sí, resulta ser todo lo que React sabe de tu aplicación y cada nodo o fibra.

**CLI:** la interfaz de línea de comandos o interfaz de línea de órdenes es un método que permite a los usuarios dar instrucciones a algún programa informático por medio de una línea de texto simple.

**Node.js:** es un entorno de ejecución de javascript que le permite al código en js ser ejecutado en nuestra computadora.

**NPM (Node Package Manager):** cuando usamos Node.js, rápidamente tenemos que instalar módulos nuevos (librerías), ya que al ser un sistema fuertemente modular viene prácticamente "vacío". Por lo tanto, para utilizar una funcionalidad de alguna librería publicada, deberemos instalar módulos adicionales. Esta operación se realiza de forma muy sencilla con esta herramienta.

## MAPA DE CONCEPTOS





# Sugar Syntax

**¿Por qué existe el  
SUGAR-SYNTAX?**



## Ejemplo en vivo

Vamos al código.

# Recordemos:

Sugar Syntax refiere a la **sintaxis agregada a un lenguaje de programación con el objetivo de hacer más fácil y eficiente su utilización.** Favorece su escritura, lectura y comprensión.

```
i = i + 1 → i++
```

# ¿Por qué existe el Sugar-Syntax?

Los lenguajes tienen una tendencia natural a evolucionar en la manera de ser escritos.

Causas principales:

- ✓ Críticas de la comunidad.
- ✓ Grado de adoptabilidad.
- ✓ Dificultad de implementar patrones de diseño comunes en otros lenguajes.

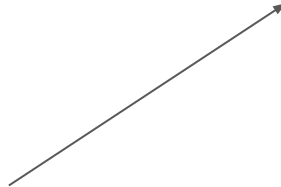
C#

```
//[modificadores de acceso] - [class] - [identificadores]
public class Customer
{
    // Fields, properties, methods and events go here...
}
```

# ¿Por qué existe el Sugar-Syntax?

```
class Animal {  
  constructor(type) {  
    this.type = 'Cat';  
  }  
  
  talk() {  
    console.log('meow');  
  }  
}
```

Para lograr esto, JS ES5 necesitaría implementar un código parecido al siguiente.



```
"use strict";  
  
function _instanceof(left, right) { if (right != null && typeof Symbol !==  
"undefined" && right[Symbol.hasInstance]) { return !!right[Symbol.hasInstance](left);  
} else { return left instanceof right; } }  
  
function _classCallCheck(instance, Constructor) { if (!_instanceof(instance,  
Constructor)) { throw new TypeError("Cannot call a class as a function"); } }  
  
function _defineProperties(target, props) { for (var i = 0; i < props.length; i++) {  
var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false;  
descriptor.configurable = true; if ("value" in descriptor) descriptor.writable =  
true; Object.defineProperty(target, descriptor.key, descriptor); } }  
  
function _createClass(Constructor, protoProps, staticProps) { if (protoProps)  
_defineProperties(Constructor.prototype, protoProps); if (staticProps)  
_defineProperties(Constructor, staticProps); return Constructor; }  
  
var Animal = /*#__PURE__*/function () {  
  function Animal(type) {  
    _classCallCheck(this, Animal);  
  
    this.type = 'Cat';  
  }  
  
  _createClass(Antimal, [{  
    key: "talk",  
    value: function talk() {  
      console.log('meow');  
    }  
  }]);  
  
  return Animal;  
}();
```

# Ejemplos de Sugar-Syntax

Implementando los patrones y sugars adecuados podemos **mejorar la legibilidad y pragmatismo** de nuestro código:

```
> const condition = true;
let result = null;
if (condition) {
  result = 'correct';
} else {
  result = 'incorrect';
}

console.log(`This is ${result}`);
This is correct VM697:9
```

Con **ternary operator**:

```
> const condition = true;
console.log(`This is ${condition ? 'correct' : 'incorrect'}`);
This is correct VI
```

# Otros ejemplos

- ✓ Spread operator

`[a, ...arr]`

- ✓ Propiedades dinámicas

`{ foo: "bar", [ "baz" + id ]: 42 }`

- ✓ Deep matching

`var { a: val } = { a: 2 }`

- ✓ Asignación en desestructuración

`var [ a = 1, b = 2, c = 3, d ] = [ 4, 5 ]`



# Arrow Functions

javascript

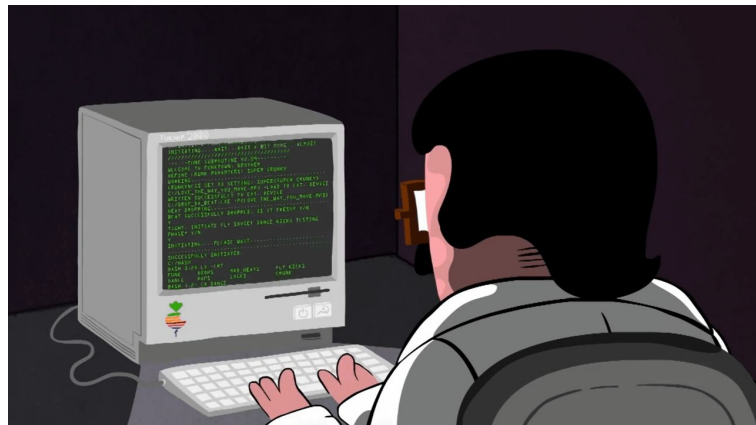
// Forma tradicional

```
function sum(a, b) {  
  return a + b;  
}
```

// Con syntax sugar

```
const sum = (a, b) => a + b;
```

- ✓ Sin llaves o con parentesis => return implicito
- ✓ Puedo asignar o no la función a una variable



# Literales de plantilla

```
javascript Copy code

// Forma tradicional
const name = "Juan";
const greeting = "Hola, " + name + "!";

// Con syntax sugar
const name = "Juan";
const greeting = `Hola, ${name}!`;
```

- ✓ Puedo concatenar strings con variables de forma rápida

# Desestructuración

javascript

// Forma tradicional

```
const person = { name: "Juan", age: 25 };
```

```
const name = person.name;
```

```
const age = person.age;
```

// Con syntax sugar

```
const person = { name: "Juan", age: 25 };
```

```
const { name, age } = person;
```

javascript

// Forma tradicional

```
const numbers = [1, 2, 3, 4, 5];
```

```
const firstNumber = numbers[0];
```

```
const secondNumber = numbers[1];
```

// Con syntax sugar

```
const numbers = [1, 2, 3, 4, 5];
```

```
const [firstNumber, secondNumber] = numbers;
```

- ✓ Puedo “desestructurar” objetos
- ✓ También puedo hacerlo con arrays



# Encuesta sobre Syntax Sugar

Por encuestas de Zoom

¡Despejemos DUDAS!

Repasamos los conceptos con una simple encuesta. **Puedes elegir más de uno.**

# POLYFILLS y la retrocompatibilidad

**¿Por qué necesito ser  
retrocompatible?**

# Ser retrocompatible

Cuando desarrollemos y pensemos la **experiencia** de nuestras aplicaciones, es importante tener en cuenta qué distribución tiene hoy el mundo, así como nuestro **target** de usuarios.



# Can I use

# fetch

[Settings](#)

50 results found

## Fetch - LS

Usage

% of all users



Global

95.43% + 0.07% = 95.5%

A modern replacement for XMLHttpRequest.

Current aligned


Usage relative

Date relative

Apply filters

Show all



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android
		2-33	4-39		10-26						
		<sup>1 4</sup> 34-38 	<sup>2</sup> 40		<sup>2</sup> 27						
	12-13	<sup>4</sup> 39	<sup>2 3</sup> 41	3.1-10	<sup>2 3</sup> 28	3.2-10.2					
6-10	14-83	40-78	42-83	10.1-13	29-68	10.3-13.3		2.1-4.4.4	12-12.1		
11	84	79	84	13.1	69	13.5	all	81	46	84	68
		80-81	85-87	14-TP		14.0					

<https://caniuse.com/>

**CODERHOUSE**



**Una historia de  
retrocompatibilidad:  
El mundo de los  
“sumadores” es invadido  
por los “multiplicadores”**

# Érase una vez...

El **antiguo mundo** que siempre fue conocido por saber la existencia de los números y su capacidad para sumarlos...

# Consideremos la situación...

Al llegar los multiplicadores, empezaron a colonizar y establecieron la multiplicación como **nuevo método** de operar.

# Consideremos la situación...

Obligados por la fuerte falta de inclusión que generó esto, apareció 'Francis **Polyfill**' con una gran idea...

"¡Sabemos que ustedes no pueden multiplicar, pero no os preocupéis!"  
Si les cuesta multiplicar... **pueden tratar lo siguiente:** "¡simplemente sumen!"

# Consideremos la situación...

Si nosotros hacemos **10 x 2**, y ustedes no saben hacerlo, simplemente hagan lo que hicieron siempre: ¡sumen!

$$2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 = 20$$

# Resultado

Entonces se entendieron y se integraron. Crearon una **manera de resolver un problema nuevo** con los recursos que ya tenían, y nadie quedó excluido de multiplicar.

# Conclusión

Los **polyfills** nos permiten hacer nuestra aplicación compatible con navegadores antiguos, que no admiten de forma nativa alguna nueva funcionalidad

# ¿Cómo se integra un POLYFILL?

Ejemplo: core-js

[zloirock/core-js: Standard Library](https://github.com/zloirock/core-js: Standard Library)

```
npm install --save core-js@3.6.5
```

```
import 'core-js'; // <- at the top of your entry point

Array.from(new Set([1, 2, 3, 2, 1]));           // => [1, 2, 3]
[1, [2, 3], [4, [5]]].flat(2);                  // => [1, 2, 3, 4, 5]
Promise.resolve(32).then(x => console.log(x)); // => 32
```





# Syntax sugar y Polyfills

Vamos a practicar lo aprendido hasta ahora

Duración: 15 minutos



ACTIVIDAD EN CLASE

# Syntax sugar & Polyfills

## Descripción de la actividad.

Implementar una función para obtener los casos:

- getCantidadAlumnosAprobados
- getCantidadAlumnosDesaprobados
- getBestStudent

Ver si nuestro código tiene syntax sugar.

[github](#)

## Descripción de la actividad.

Implementar un polyfills del método de los arrays utilizado.

Imagina que el método FIND/FILTER/SORT de los arrays es implementado de forma correcta sólo por muy pocos navegadores. Sin embargo, tú necesitas su funcionalidad dentro de tu aplicación.

En este escenario, necesitas implementar un polyfills del método para asegurar que todos los usuarios (no importa el browser que usen) puedan utilizar tu aplicación.

Cuentas con 15 minutos para resolver esta actividad.



# Break

¡10 minutos y volvemos!

# Bundling con WEBPACK

# Webpack

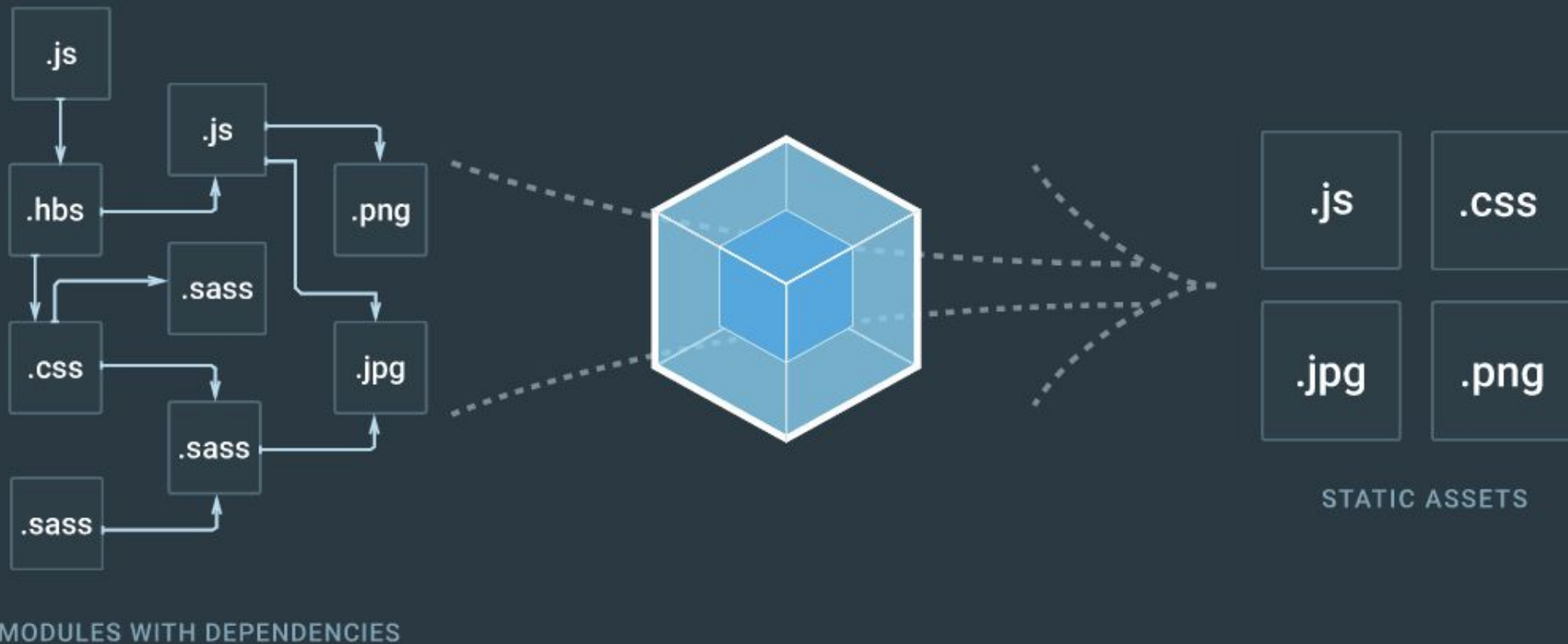


# webpack

Webpack **es un module bundler o empaquetador de módulos** que nació a finales de 2012, y en la actualidad es utilizado por miles de proyectos de desarrollo web Front-End.

Incluido desde React o Angular hasta en el desarrollo de aplicaciones conocidas como Twitter, Instagram, PayPal, o la versión web de Whatsapp.

# Transformación de los módulos en Webpack



# ¿Cómo funciona?

Podemos tener, por ejemplo, un módulo JS que vaya a depender de otros módulos .js, con imágenes en diferentes formatos como JPG o PNG. O estar utilizando algún preprocesador de CSS, como puede ser SASS, Less y Stylus.

Webpack recoge todos estos módulos y los transforma a assets que puede entender el navegador, como por ejemplo archivos JS, CSS, imágenes, videos, etc.

# ¿Cómo nos afecta en nuestro desarrollo?

Internamente está incluido en la aplicación generada por **create-react-app**.

Importante: **el equipo de react es quien se encarga de mantener estas configuraciones actualizadas.**

Podemos modificarlas, pero para eso necesitamos realizar un **eject**.

```
> react-scripts eject
```

```
NOTE: Create React App 2+ supports TypeScript, Sass, CSS Modules and
```

```
? Are you sure you want to eject? This action is permanent. (y/N) █
```





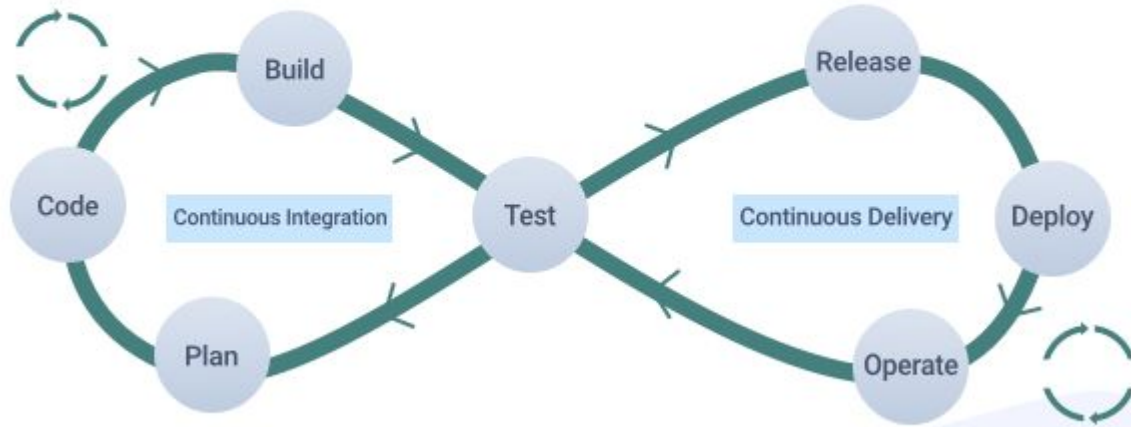
## Ejemplo en vivo

Vamos al código.

# IT brackets

## CI/CD

(Continuous Integration/Continuous Delivery)

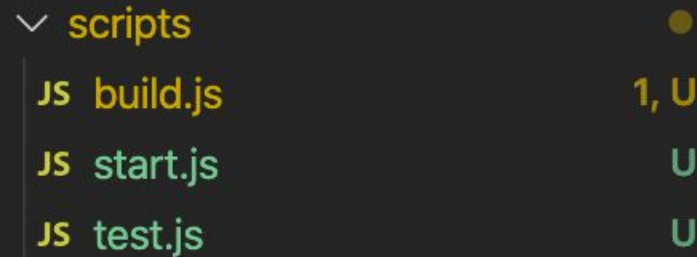


[LINK](#)

# ¿EJECT?

Es una acción **permanente**, que permite tener un control más específico del bundling, a costa de que de ahora en adelante tendremos que encargarnos de mantenerlo.

```
npm run eject
```



```
▼ scripts  
JS build.js 1, U  
JS start.js U  
JS test.js U
```

# Costo y alternativas

En algunas oportunidades, cuando tengamos más experiencia, nos puede dar **más flexibilidad**, pero **no siempre** es el caso. Hay algunos proyectos que dan alternativas, como **rewired**, pero el resumen es:



**Dan Abramov**  
@dan\_abramov

Replying to [@AdamRackis](#)

"Stuff can break" — Dan Abramov

7:57 PM · Sep 28, 2018 · [Twitter Web App](#)

*"Las cosas se pueden romper..."*

# Transpiling

# ¿Qué es el Transpiling?



# Transpiling

Es el proceso de **convertir código** escrito en un lenguaje, a **su representación en otro lenguaje**. Usualmente extienden o simplifican la escritura del lenguaje, o representación original.

- ✓ Implementan un proceso similar conceptualmente al **pollyfilling**.
- ✓ Logran niveles de simetricidad y simbiosis con el lenguaje original.

JSX



**¿Qué es y por qué lo  
usamos?**

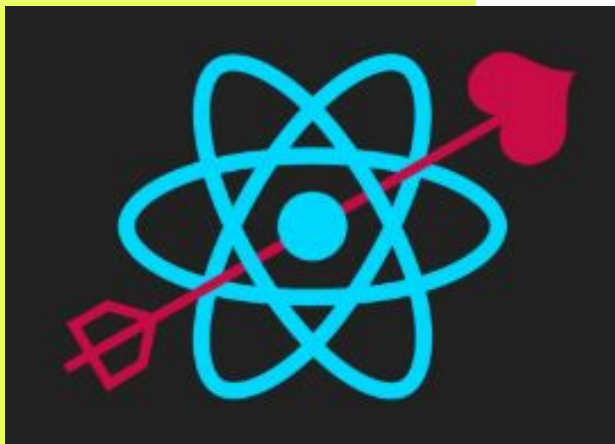
# Javascript xml

**JSX** es una extensión de sintaxis de Javascript que se parece a HTML

Oficialmente, es una **extensión que permite hacer llamadas a funciones y a construcción de objetos**. No es ni una cadena de caracteres, ni HTML.

The logo for JSX, featuring the letters 'JS' in a bold, dark blue font on a yellow background, and the letter 'X' in a bold, white font on a purple background.

# JSX



**JSX es una extensión de Javascript, no de React.**

Esto significa que **no hay obligación de utilizarlo**, pero es **recomendado** en el sitio web oficial de React.

# Funcionamiento y características

# ¿Cómo funciona?

JSX se transforma en código JavaScript.

Esto nos da algunas ventajas, como ver errores en tiempo de compilación, asignar variables, retornar métodos, etc.

```
<div className="active">Hola Coders</div>  
  
React.createElement('div', { className: 'active'}, 'Hola Coders');
```

# Styling en JSX

Es posible definir y utilizar estilos inline en JSX, solo necesitamos convertirlos por convención:

border-color => borderColor

padding-top => paddingTop

'10px' => 10 (no es necesario el px)

```
let styles = {  
  borderColor: '#999'  
};  
  
const jsx = (  
  <div style={styles}>  
    Hola Coders  
  </div>  
)
```

# Inline styles en JSX

Los mismos estilos se pueden configurar **inline** en JSX, solo necesitamos usar doble llave `{{ }}`,

- ✓ La **primera** llave para avisar que se agregará un **objeto** en js.
- ✓ La **segunda** llave para empezar a escribir el objeto en sí.

```
const Salute = () => <p style={{ marginLeft: 15}}>Hello</p>
```

# Reglas generales

- ✓ Los elementos deben ser balanceados. Por cada apertura debe haber un cierre.

`<img src="">` Mal

`<img src=""></img>` Es mejorable

- ✓ Si el elemento no tiene hijos, debe idealmente ser auto-cerrado

`<img src="" />` Ideal



# Reglas generales

Class es palabra reservada, en su lugar usar className.

`<img src="" class="my-class" />` Mal

```
<img src="" className="my-class" /> Ok
```

```
return (
  <h1 className='large'>Hello World</h1>
);
```





## Ejemplo en vivo

Vamos al código.

# ¡No olvidemos!

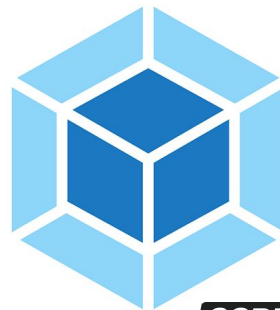
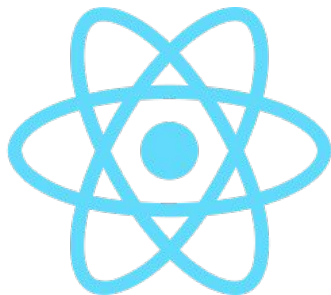
En JSX se utilizan tanto los estilos como los eventos estándar del DOM, como **onclick**, **onchange**, **onkeydown**, etc. pero utilizando **camelCase**: `onClick`, `onChange`, `onKeyDown` / `marginTop`, `paddingBottom`, etc.



# ¡No olvidemos!

A medida que nuestra aplicación va creciendo y tenemos componentes más grandes que manejan distintos eventos, JSX nos va a ayudar mucho a agilizar y organizar nuestro desarrollo de componentes.

Para poder utilizar JSX en nuestra aplicación, debemos tener instalado **Webpack** que en nuestro caso viene incluido con **create-react-app**.

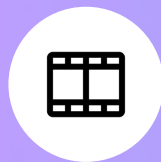




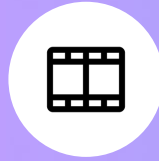
# Primera pre-entrega

En la clase que viene se entregará la primera parte del Proyecto final, que **nuclea temas vistos entre las clases 1, 2, 3 y 4**.  
Recuerda que tendrás 7 días para subirla en la plataforma.

¿Preguntas?



**¿Quieres saber más?**  
**Te dejamos material  
ampliado de la clase**



**Nuestro GITHUB**  
LINK





MATERIAL AMPLIADO

# Recursos multimedia

## Artículos

- ✓ [React.js Essentials \(1 ed.\). EEUU, Packt.](#) | Fedosejev, A. (2015)
- ✓ [ReactJS by Example \(1 ed.\). EEUU, Packt.](#) | Amler (2016)

## Articulos

- ✓ [ReactJS Cookbook \(1 ed.\). EEUU, Packt.](#) | Stein, J. (2016)
- ✓ <https://caniuse.com> | Can I use

# Resumen de la clase hoy

- ✓ Sugar syntax.
- ✓ Retrocompatibilidad.
- ✓ Webpack.
- ✓ JSX.

**Opina y valora**  
**esta clase**

**Muchas gracias.**

**#DemocratizandoLaEducación**