



Clase 03. React JS

JSX Y WEBPACK



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

SUGAR SYNTAX

***¿Por qué existe el
sugar-syntax?***

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}`);
```

con **ternary operator**

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


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]`

VAMOS AL CÓDIGO



CODER HOUSE

POLYFILLS Y LA RETROCOMPATIBILIDAD

***¿Por qué necesito 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						
		^{1 4} 34-38	² 40		² 27						
	12-13	⁴ 39	^{2 3} 41	3.1-10	^{2 3} 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					

***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...

Consideremos la situación

“¡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!

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

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)

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

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

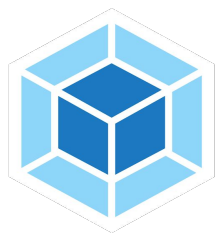
Array.from(new Set([1, 2, 3, 2, 1]));           // => [1, 2, 3]
[1, [2, 3], [4, [5]]].flat(2);                  // => [1, 2, 3, 4, 5]
Promise.resolve(32).then(x => console.log(x)); // => 32
```

BREAK



CODER HOUSE

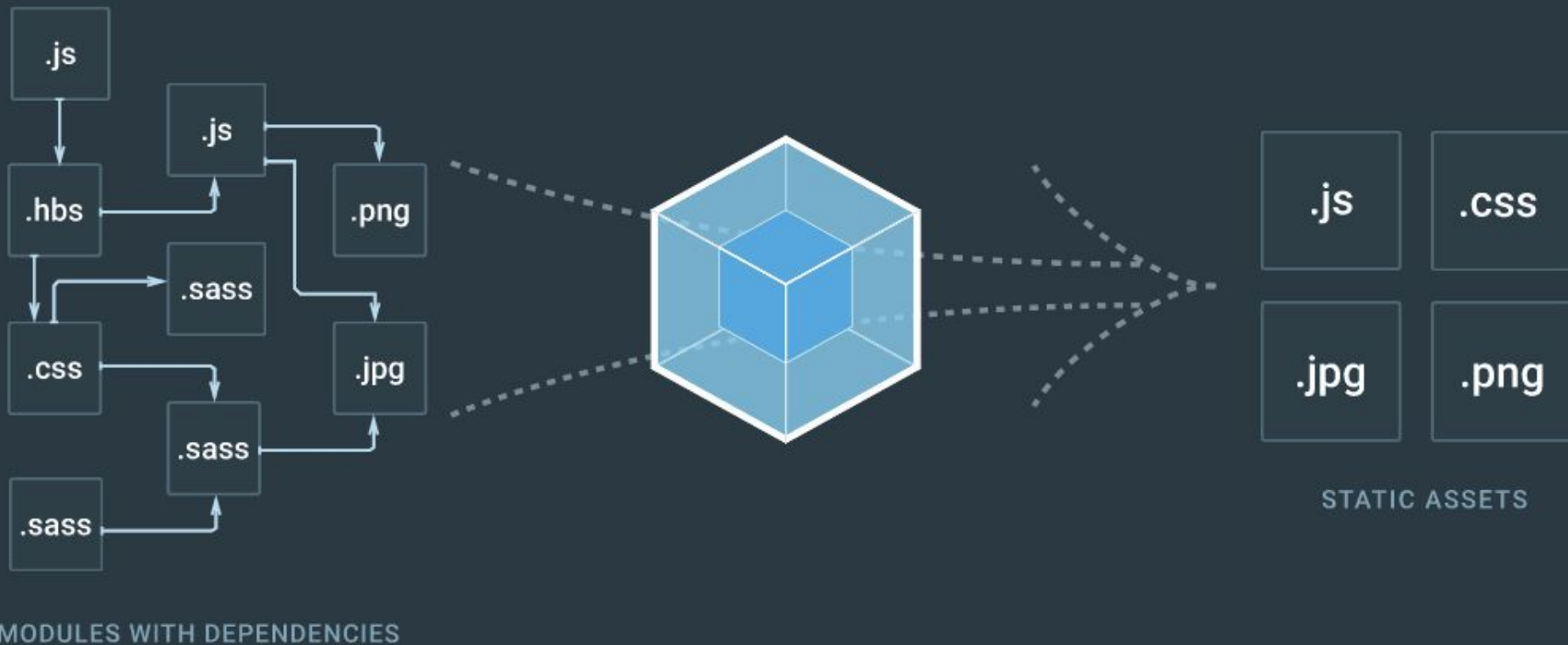
BUNDLING CON WEBPACK



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
```

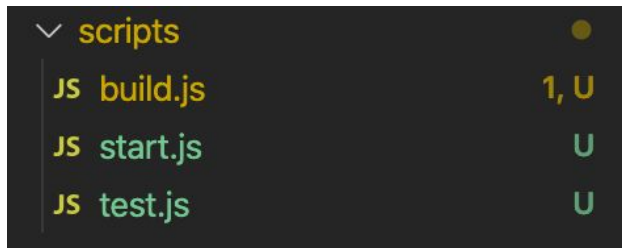
```
NOTE: Create React App 2+ supports TypeScript, Sass, CSS Modules and
```

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

¿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
```



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 les traigo esta noticia:



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?***



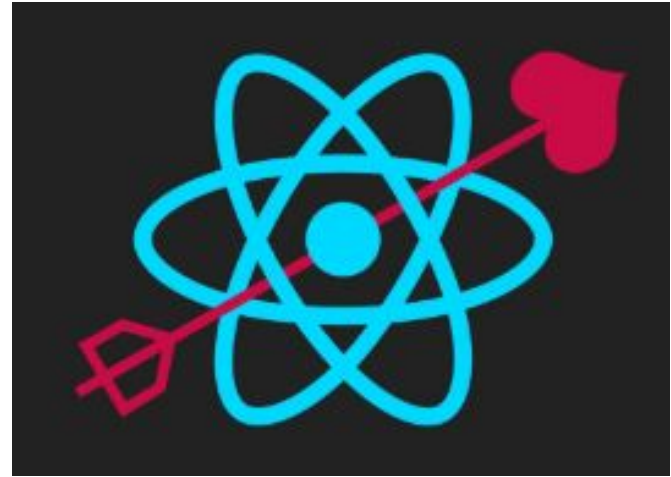
javascript **x**ml

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.

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

```
07  
08  
09 <div className="active">Hola Coders</div>  
10  
11  
12 React.createElement('div', { className: 'active'}, 'Hola Coders');  
13  
14
```

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

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)

```
104  
105  
106 let styles = {  
107   |   borderColor: '#999'  
108   | };  
109  
110 const jsx = (  
111   |   <div style={styles}>  
112   |     Hola Coders  
113   |   </div>  
114   | )
```

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.

`` Mal

`` Es mejorable

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

`` Ideal

Reglas generales

Class es palabra reservada, en su lugar usar className



Mal

`` Ok

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



VAMOS AL CÓDIGO



CODER HOUSE

¡No olvidemos!

En JSX se utilizan tanto los estilos como los eventos estándar del DOM, como

onclick, **onchange**,

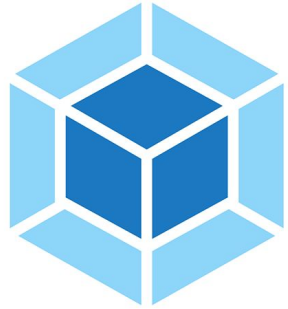
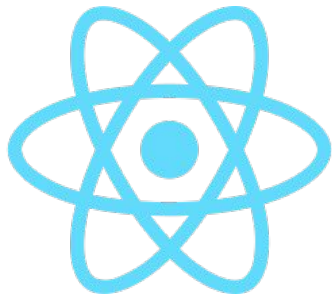
onkeydown, etc. pero

utilizando **camelCase**:

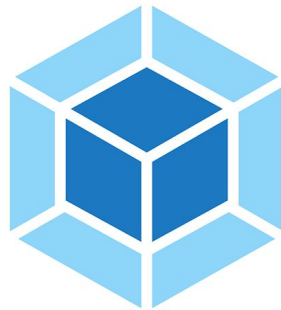
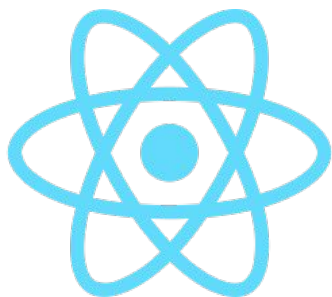
onClick, onChange,
onKeyDown / marginTop,
paddingBottom, etc.



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**



¿PREGUNTAS?





Menú e-commerce

En el directorio de tu proyecto, crea una carpeta dentro de src llamada components que contenga a NavBar.js para crear una barra de menú simple

Formato de entrega: carpeta comprimida con los archivos del proyecto

Tiempo: 30 minutos



RECURSOS DE LA CLASE

- Fedosejev, A. (2015). React.js Essentials (1 ed.). EEUU, Packt.
- Amler (2016). ReactJS by Example (1 ed.). EEUU, Packt.
- Stein, J. (2016). ReactJS Cookbook (1 ed.). EEUU, Packt.
- <https://caniuse.com>

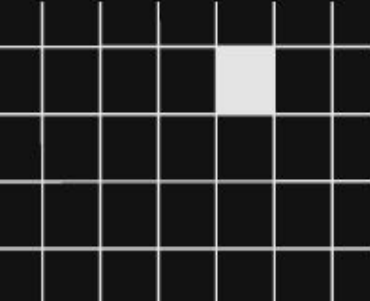
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Sugar syntax
 - Retrocompatibilidad
 - Webpack
 - JSX
- 



OPINÁ Y VALORÁ ESTA CLASE