



# Curso de React y React Native

## Clase 01



# Presentarnos.

*Nombre | Qué hago | Qué espero del curso*



# Agenda de la clase



# Agenda

- Antes de comenzar...
- Introducción a React.
- JSX.
- Componentes.



Antes de comenzar...



# Antes de comenzar...

- Instalar **Google Chrome** (<https://www.google.com/chrome/>).
- Instalar Microsoft **Visual Studio Code** (<https://code.visualstudio.com/>).
  - Es posible que la instalación requiera que se instale el programa [Git](#).
  - Otra opción: Atom (<https://atom.io/>).
- Instalar **Node.js** (<https://nodejs.org/en/>).
- Crearse una cuenta de **GitHub** (<https://github.com/>).
- Unirse a **Slack** (<https://slack.com/>).
  - Slack es una herramienta de comunicación ideal para equipos de trabajo.
  - Recibirán una invitación a su correo electrónico.
  - Link: <https://hack-academy.slack.com>.
  - Recomendado: descargarse app de Slack para smartphone y desktop.

# Slack – Introducción



Tip: una buena forma de probar Slack es escribirse a ustedes mismos.

- Herramienta de comunicación ideal para equipos de trabajo.
- Las conversaciones se organizan en **canales**. Ejemplo:
  - `#react-XXXX-material`: Para compartir material del curso como diapositivas o ejercicios.
  - `#react-XXXX-dudas`: Para consultar dudas sobre ejercicios o temas teóricos.
  - `#react-XXXX-general`: Para compartir información en general sobre el curso.
  - `#random`: Para compartir cualquier tipo de información.
- Permite **pegar código** en los mensajes escribiendo el mismo entre comillas (simples o triples) de este tipo: ``` (acento grave o backtick). [Más info](#).
- Permite encontrar información mediante potente **buscador**.
- Permite enviar mensajes directos (privados) entre dos o más usuarios.



# Antes de comenzar... ¿qué deberían saber?

La siguiente es una lista de temas de JavaScript que el alumno debería dominar antes de anotarse al curso de React:

- Tipos de datos.
- Operadores.
- If/Else.
- For/While Loops.
- Funciones.
- Callbacks.
- Eventos.
- Arrays (Arreglos).
- Objetos.
- Scope global y local.

Parámetros por referencia y por valor.





# Antes de comenzar... Tips generales

- ¡Hagan muchas **preguntas**! (no sientan vergüenza).
- "**Googleen**" mucho. También consulten [Stack Overflow](#).
- [En general] No copien/peguen código salvo que entiendan o sepan lo que están copiando.
- Sean **pacientes** consigo mismos. Para la mayoría no es fácil al principio.
- Intenten **ayudarse** entre ustedes. Enseñar es una gran forma de aprender.



# Pedidos

- Comenzar las clases en hora.
- Cuidar la limpieza del salón.
- Cuidar la limpieza de los baños.
- Evitar traer líquidos al salón. Mate OK.
- No tirar vasos con líquido en las papeleras.
- Usar el grupo de WhatsApp con moderación.



# ¿Qué es React?



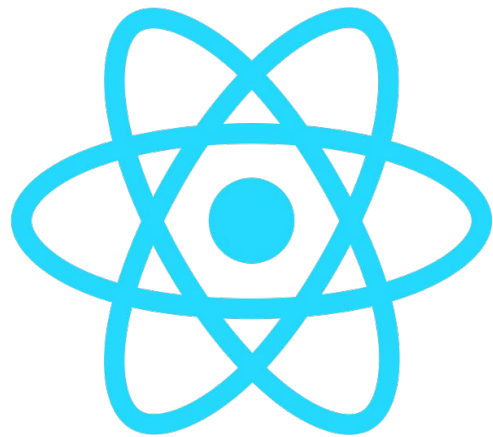
# ¿Qué es React? (1)

React es una **librería Javascript** para construir interfaces de usuario (UI), desarrollada y mantenida por Facebook, pero Open Source.

Cuenta con una gran comunidad de desarrolladores.

React está basado en el desarrollo de componentes, pequeños bloques que se pueden componer para poder lograr interfaces de usuario más complejas.

Documentación: [link](#).





## ¿Qué es React? (2)

React está diseñado de tal forma que, aprendiendo sus conceptos, sea muy simple desarrollar para diferentes ambientes, como web mediante **ReactDOM** o para plataformas móviles mediante **React Native**.

React Native cuenta con la ventaja de poder escribir una aplicación tanto para **Android** como para **iOS** con básicamente el mismo código (JavaScript), pero a diferencia de otros desarrollos “Cross-Platform”, el resultado final son aplicaciones **nativas**.



# ¿Por qué React? (1)

React es Declarativo. Veamos el siguiente árbol de componentes. ¿Podemos leerlo y entender qué realiza nuestra aplicación?

```
<MyApp>
  <Header>
    <Menu />
  </Header>
  <Main>
    <Products />
  </Main>
  <Footer />
</MyApp>
```



## ¿Por qué React? (2)

React está basado en **componentes**, por lo construir una aplicación en React se puede comparar a jugar con Legos, donde cada pieza de Lego es un componente (desarrollado por nosotros, otro del equipo o de una librería de terceros).

Cada **componente** debería ser lo más independiente posible, para favorecer la reutilización, pero también fácil de conectar a los demás. Cuanto más chicos, más fácil de mantenerlos y entenderlos.



# ¿Por qué React? (3)

- Se puede agregar a aplicaciones existentes de forma sencilla.
- Creada por Facebook y utilizada en sus propios productos.
- Tiene un gran ecosistema, compuesto por muchas librerías y frameworks auxiliares para utilizar en conjunto con React. Ej: Redux, React Router, Next.js y otras.
- Grandes jugadores la utilizan, y aportan a la comunidad y el ecosistema: Airbnb, Netflix, Apple, Instagram, Paypal, etc.
- Permite la creación de aplicaciones complejas, con UIs dinámicas y que manejan muchos datos.





# Un componente de React

Los componentes en React son funciones. Reciben un input y devuelven un output, determinístico, expresado en sintaxis **JSX** (de la que se hablará posteriormente).

```
import React from 'react';
import ReactDOM from 'react-dom';

const HelloMessage = ({name}) => <div>Hola {name}</div>;

ReactDOM.render(
  <HelloMessage name="John" />,
  document.getElementById('root')
);
```

JSX le llamamos a poder usar tags de HTML en el código Javascript.



# DOM y Virtual DOM en React



# Manejo del DOM

El **DOM** (Document Object Model) es el árbol de elementos HTML que toda página web tiene. En el desarrollo web es común trabajar creando, modificando o borrando elementos del DOM (utilizando jQuery por ejemplo).

En React el DOM es manejado internamente y nosotros sólo tenemos que definir ***Componentes***. React es luego el encargado de interpretar qué elementos del DOM fueron modificados, para actualizar sólo esos elementos.

Para esto, internamente, utiliza algo llamado **Virtual DOM**, propio de React, que es una copia del DOM pero mucho más liviana y rápida.

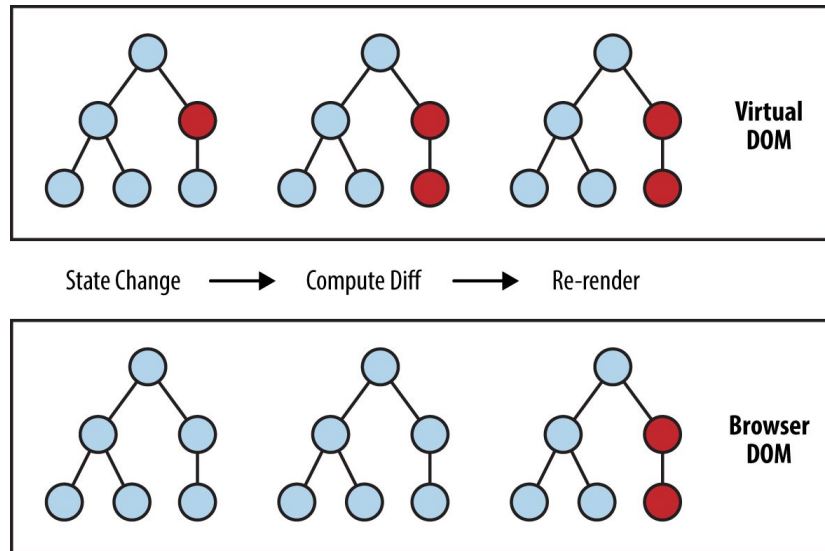


# Virtual DOM (1)

Esta es una de las grandes ventajas de React: la responsabilidad del programador es solo **mantener el estado**.

React luego se encarga de definir cuál es el set mínimo viable de operaciones a ejecutar sobre el DOM para reflejar dicho cambio de estado.

En React no se actualiza el DOM manualmente.





# Virtual DOM (2)

Esta demo la podemos encontrar en este [link](#).

Hello JS

Tue Dec 27 2016 13:02:43  
GMT-0800 (PST)

Hello React

Tue Dec 27 2016 13:02:43  
GMT-0800 (PST)

```
Elements Console Sources Network Timeline Pro
<!DOCTYPE html> == $0
<html>
  <head>...</head>
  <body>
    <div id="js">
      <div class="demo">...</div>
    </div>
    <div id="react">
      <div data-reactroot="" class="demo">
        <!-- react-text: 2 -->
        "Hello React"
        <!-- /react-text -->
        <input>
        <p>Tue Dec 27 2016 13:02:43 GMT-0800 (PST)</p>
      </div>
    </div>
    <script src="script.js" charset="utf-8"></script>
  </body>
</html>
```



# Introduciendo JSX

# JSX

Pueden probar traducir cualquier código JSX a JavaScript "normal" en [Babel](#).



```
const element = <h1>¡Hola Mundo!</h1>;
```

Esto no es HTML ni un string. Es JSX, que es una **extensión a la sintaxis de JavaScript**.

Se utiliza para definir elementos de React.

JSX produce elementos de React, que en realidad son simplemente objetos en JavaScript!

```
var element = <h1>¡Hola Mundo!</h1>;
```

Se traduce en

```
var element = React.createElement(  
  "h1",  
  null,  
  "¡Hola Mundo!"  
);
```



# Expresiones en JSX

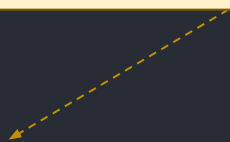
Dentro de los tags de JSX es posible utilizar cualquier expresión JavaScript, ya sea una cuenta matemática, o evaluar una función.

```
const formatName = user => user.firstName + ' ' + user.lastName;
```

```
const user = {  
  firstName: 'Josefina',  
  lastName: 'Pérez',  
};
```

```
const element = <h1>Hola {formatName(user)}!</h1>;
```

Cuando este elemento se muestre al usuario, será con "Hola Josefina Pérez".





# Atributos en JSX



A los tags de JSX se les puede especificar atributos, al igual que sus tags equivalentes en HTML. La diferencia es que en JSX se escriben utilizando nomenclatura camelCase:

- `tabindex` se convierte en `tabIndex`
- `onclick` se convierte en `onClick`

```
const element = <div tabIndex="0"></div>;
```

Un caso a tener en cuenta: En Javascript `class` es una palabra reservada, por lo que para proporcionar una clase, en JSX usamos `className`.

```
const element = (  
  <h1 className="greeting">  
    ¡Hola Mundo!  
  </h1>  
)
```

La clase `greeting` está definida en otro archivo CSS. Por ejemplo:

```
.greeting {  
  color: "red"  
}
```



# Componentes en React



# Utilizando Componentes

Un elemento es el bloque más chico en React y describe qué mostrar en pantalla.

```
const element = <h1>¡Hola Mundo!</h1>;
```

Para mostrar un elemento en pantalla, se necesita un nodo del DOM que se utilizará como "raíz". En general, una aplicación construida en React sólo tendrá un nodo, pero si se agrega React a una aplicación existente, puede haber varios nodos "raíz". Utilizando `ReactDOM.render` se podrá indicarle a React dónde renderizar el componente raíz.

```
const element = <h1>¡Hola Mundo!</h1>;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
)
```

En nuestro proyecto, debe haber un html que dentro tenga un `div` con id "root" para que esto funcione.



# Componentes y Props (1)

Los componentes permiten separar la UI en pedazos chicos, independientes y reutilizables. Conceptualmente, los componentes son **funciones** JavaScript; reciben parámetros (*props*) y retornan lo que se debe mostrar en pantalla (utilizando JSX).

```
function WelcomeMessage(props) {  
  return <h1>Hola {props.name}</h1>;  
}
```

```
class WelcomeMessage extends React.Component {  
  render() {  
    return <h1>Hola {this.props.name}</h1>;  
  }  
}
```

Esto es llamado “class component”, y está utilizando las clases de Javascript introducidas en ES6.

Estos dos componentes son equivalentes desde el punto de vista de React, aunque el **class** component tiene algunas características diferentes que veremos más adelante. Por ahora, notar que desde uno utilizamos `this.props` y desde el otro solo `props`.



# Componentes y Props (2)

Hasta ahora se vieron elementos de React que utilizan sólo tags de HTML, pero se pueden crear elementos a partir de los Componentes que nosotros mismos definimos.

```
// Definimos el Componente
function WelcomeMessage(props) {
  return <h1>Hola, {props.name}</h1>;
}
```

```
// Utilizamos el Componente
var element = <WelcomeMessage name="María" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Cuando React encuentra un elemento definido por nosotros, le pasa los atributos definidos en JSX al componente como **props**.



# Componentes y Props (3)

Lo que está sucediendo en el bloque de código anterior es:

- Se llama a `ReactDOM.render()` con el elemento `<WelcomeMessage name="María" />`
- React llama al componente `WelcomeMessage` con las props `{ name: "María" }`
- Nuestro componente `WelcomeMessage` retorna `<h1>Hola María</h1>`
- React DOM eficientemente actualiza el DOM para que coincida con `<h1>Hola María</h1>` dentro del div con `id="root"`.



# Componer Componentes

Como se mencionó, la idea de los *componentes* en React es poder componerlos.

```
function WelcomeMessage(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
function App() {  
  return (  
    <div>  
      <WelcomeMessage name="María" />  
      <WelcomeMessage name="Marta" />  
      <WelcomeMessage name="Lucía" />  
    </div>  
  );  
}  
  
ReactDOM.render(<App />, document.getElementById('root'));
```

Como regla, debemos nombrar nuestros componentes comenzando con mayúscula (Pascal Case), para que React los distinga con los tags propios de HTML.

Como queremos devolver más de un elemento, una opción es envolverlos en un `<div>`.

# Componer Componentes



Estas dos formas también son válidas para retornar una lista de elementos, sin necesidad de crear otro nodo del DOM.

```
function WelcomeMessage(props) {  
  return <h1>Hola {props.name}</h1>;  
}  
  
function App() {  
  return [  
    <WelcomeMessage name="María" />,  
    <WelcomeMessage name="Marta" />,  
    <WelcomeMessage name="Lucía" />,  
  ];  
}  
  
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
);
```

```
function WelcomeMessage(props) {  
  return <h1>Hola {props.name}</h1>;  
}  
  
function App() {  
  return (  
    <>  
      <WelcomeMessage name="María" />  
      <WelcomeMessage name="Marta" />  
      <WelcomeMessage name="Lucía" />  
    </>  
  );  
}  
  
ReactDOM.render(<App />,  
  document.getElementById('root'));
```





# Herramienta extra – Developers Tools

En React existen las “Developer Tools”.

Se pueden instalar como extensión de [Chrome](#) o [Firefox](#).

Nos permiten inspeccionar árboles de componentes React.

```
▼<Game>
  ▼<div className="game">
    ▼<div className="game-board">
      ▼<Board>
        ▼<div>
          <div className="status">Next player: X</div>
          ▼<div className="board-row">
            ▶<Square index=0>...</Square>
            ▶<Square index=1>...</Square>
            ▶<Square index=2>...</Square>
          </div>
          ▼<div className="board-row">
            ▶<Square index=3>...</Square>
            ▶<Square index=4>...</Square>
            ▶<Square index=5>...</Square>
          </div>
          ▼<div className="board-row">
            ▶<Square index=6>...</Square> == $r
            ▶<Square index=7>...</Square>
            ▶<Square index=8>...</Square>
          </div>
        </div>
      </Board>
    </div>
  </div>
  ▼<div className="game-info">
    <div />
    <ol />
  </div>
</Game>
```



# Ejercicios



# Instalaciones previas

- Para trabajar con React es necesario tener instalado **Node.js**:  
<https://nodejs.org/en/>.
- Opcionalmente, podrán instalar **yarn** que para usar como sustituto de **npm** (para instalar paquetes a nuestros proyectos).
- Lo otro que se necesita es un editor de texto. Se podrá utilizar **Visual Studio Code**, Atom, Sublime o algún otro.
  - En el caso de Sublime es necesario instalar además un plugin para que interprete JSX. Atom y Visual Studio Code lo traen incluido. Recomendamos usar Visual Studio Code.



# Proyecto base

Utilizaremos `create-react-app`, la opción más usada y la mejor opción para el 99% de los proyectos de react.

1. Ejecutar el siguiente comando para crear un proyecto nuevo. Este comando se usará cada vez que se cree un nuevo proyecto:

```
$ npx create-react-app mi-proyecto
```

2. Luego, desde la consola, “pararse” en la carpeta del nuevo proyecto e iniciarlo.

```
$ cd mi-proyecto
```

```
$ npm start
```

En caso de que el browser no se abra de forma automática, entrar a <http://localhost:3000/>.



# Ejercicio

1. Crear un nuevo proyecto React llamado `mi-proyecto` (o como quieran)
2. Abrir dicha carpeta en VSC (como un **proyecto**).
3. Desde la consola, parados en la carpeta del proyecto, iniciar el mismo (`npm start`).
4. Revisar el archivo `src/App.js`, probar hacer algunos cambios en los textos. Guardar y revisar en el navegador los cambios. (Notar que no es necesario recargar el navegador para ver los cambios)

# Ejercicio



1. En `App.js` crear un componente llamado `Badge` que reciba como `props` los siguientes atributos: `name`, `lastName`, `imageUrl` y `nickname`.
2. Hacer que `Badge` se muestre en la pantalla y que reciba las `props` necesarias para renderizar de forma tal que:
  - a. Nombre y apellido dentro de una misma tag `<p>` separados por un espacio, haciendo uso de interpolación de strings (template strings en el material de repaso).
  - b. Nickname debe ser presentado en otra tag `<p>` como “Nickname: [Inserte nickname recibido por prop aquí]” en color gris, asignándole una clase `nickname`, la cual debe ser implementada en la hoja de estilos ya incluida por `create-react-app`.
  - c. Representar la imagen en un cuadrado de 400 x 400, aplicándole el estilo en la misma hoja de estilos.



# Ejercicio

3. Mover el componente `Badge` a un nuevo archivo llamado `Badge.js` de forma tal que pueda ser consumido desde `App.js`.
4. Modificar el componente `Badge` para que pueda recibir una prop `highlightNickname` de forma tal que el elemento que contiene el `nickname`, aparte de tener la clase `nickname`, pueda tener una clase extra de css que ponga el texto de color azul. En otras palabras: Crear nueva clase en el CSS y aplicarla o no en función del valor de la prop `highlightNickname`).
5. Usar el componente múltiples veces en `App.js` haciendo que la prop `highlightNickname` esté con valor `true`, con valor `false` o que directamente no la tenga y evaluar el comportamiento.