



FULL STACK

**Comenzamos en unos  
minutos**

---

ACADEMY  
by NUMEN

# Introduccion a Javascript ES6 - Parte 1

# Introducción



## Fundamentos de ES6

Ecmascript 6 es la última versión estandarizada de Javascript. Esta versión nos trae una serie de características que mejoran mucho la sintaxis y la experiencia con el lenguaje. Algunas de estas características las hemos nombrado durante el módulo de Javascript, y otras aún no.

Para poder trabajar con React deberemos aprender estas características.

Repasaremos las ya vistas y nos aprenderemos todas las que aún faltan a lo largo de las primeras 2 clases.

# var, let y const

ES6 nos trae 2 nuevos tipos de variables que vienen a solucionar algunos inconvenientes que presentaba la variable **var**. En esta demostración jugaremos un poco con el **console.log** viendo que imprime dependiendo de donde lo coloquemos. Esto irá arrojando varias diferencias que los sorprenderán si no están aún interiorizados con esto.

¡A JUGAR!

```
var musica = "Pop"  
console.log(musica)
```

```
{  
  var musica = "Rock"  
}
```

```
var musica = "Pop"
```

```
let musica = "Pop"  
console.log(musica)
```

```
{  
  let musica = "Rock"  
}
```

```
let musica = "Pop"
```

```
const musica = "Pop"  
console.log(musica)
```

```
{  
  const musica = "Rock"  
}
```

```
const musica = "Pop"
```

# template strings

Las cadenas templadas son una forma más moderna y práctica de lidiar con las concatenaciones de strings, donde por medio de las comillas inclinadas (backticks) podemos escribir texto de corrido, si usar los símbolos `+` y además pudiendo escribir con saltos de línea.

```
let nombre = "Academia",  
    apellido = "Numen"  
  
let concatenacionES5 = "Bienvenidos a " + nombre + " " + numen + "."
```

```
let nombre = "Academia",  
    apellido = "Numen"  
  
let concatenacionES6 = `Bienvenidos a ${academia} ${numen}.`
```

# arrow functions

Las funciones flecha son bastante parecidas a las funciones clásicas solo que con algunas cuantas ventajas.

Algo a remarcar de este tipo de funciones es que solo pueden usarse como funciones expresadas o anónimas, pero no como funciones declaradas.

```
// Función flecha clásica
const funcionFlecha1 = (a, b) => {
  return a + b
}
```

```
// Función flecha de único parámetro y única instrucción
const funcionFlecha2 = numero => numero * numero
```

# Configuración del Entorno de React



# Introducción a Node JS



Node js es un entorno de ejecución que nos permite ejecutar Javascript por fuera de un navegador. En pocas palabras, Node desata el poder de JS, dándole la posibilidad de tener las mismas capacidades que cualquier otro lenguaje de programación, tales como interactuar con nuestro sistema operativo y desarrollar aplicaciones completas.

# Instalación de Node JS



The screenshot shows the Node.js website with the following content:

- Header: node JS logo and navigation links (INICIO, ACERCA, DESCARGAS, DOCUMENTACIÓN, PARTICIPE, SEGURIDAD, NOTICIAS, CERTIFICATION).
- Text: Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.
- Alert: New security releases to be made available August 11th, 2021.
- Section: Descargar para Windows (x64)
- Two download options:
  - 14.17.4 LTS**: Recomendado para la mayoría (highlighted with a red box and arrow).
  - 16.6.1 Actual**: Últimas características.

# NPM – Node Package Manager



NPM es el Administrador de Paquetes de Node, a través del cual vamos a poder instalar las librerías que consideremos necesarias para el desarrollo de nuestra aplicación. Podemos instalarlas localmente, disponibles para usar en un proyecto en específico, o globalmente, disponibles para usar cada vez que queramos.



💧 React es una biblioteca de Javascript eficiente y declarativa que se utiliza para crear interfaces interactivas.

💧 Fue creado por el ingeniero de Facebook Jordan Walke y lanzado al mercado en 2013.

💧 Actualmente es el rey indiscutido, superando a todos sus rivales, incluyendo a AngularJS.

# Ventajas

- 💧 Alto rendimiento.
- 💧 Se puede usar tanto del lado del cliente como del servidor.
- 💧 Se integra fácilmente con otras tecnologías.
- 💧 Posee recursos abundantes ya que es mantenida por facebook y tiene una gran comunidad de contribuyentes.
- 💧 Es compatible con versiones anteriores.
- 💧 Posee una estructura fácil de mantener gracias a su arquitectura basada en componentes.
- 💧 Posee documentación multidioma.
- 💧 Maneja un flujo de datos unidireccional.

# Desventajas

- 💧 Es complejo y tiene una curva de aprendizaje elevada.
- 💧 El uso de JSX (Javascript XML) agrega una complejidad extra.
- 💧 Requiere de un gran conjunto de herramientas para funcionar de manera óptima y ser compatible con otras tecnologías.
- 💧 Tiene problemas de SEO.

**create-react-app**



💧 Create React App es un ambiente cómodo para aprender React, y es la mejor manera de comenzar a construir una nueva aplicación usando React.

## create-react-app

💧 **Create React App** configura tu ambiente de desarrollo de forma que puedas usar las últimas características de Javascript, brindando una buena experiencia de desarrollo, y optimizando tu aplicación para producción.

```
npx create-react-app my-app  
cd my-app  
npm start
```



# create-react-app

Es una herramienta de **ejecución** de paquetes de Node

Es el comando para crear una nueva app con **React**

Es el **nombre** que le quiero dar a mi nueva aplicación

Este comando nos permite **acceder** a nuestra app

Es una herramienta de **manejo** de paquetes de Node

Este comando nos permite **iniciar** nuestra app

```
npx create-react-app my-app
```

```
cd my-app
```

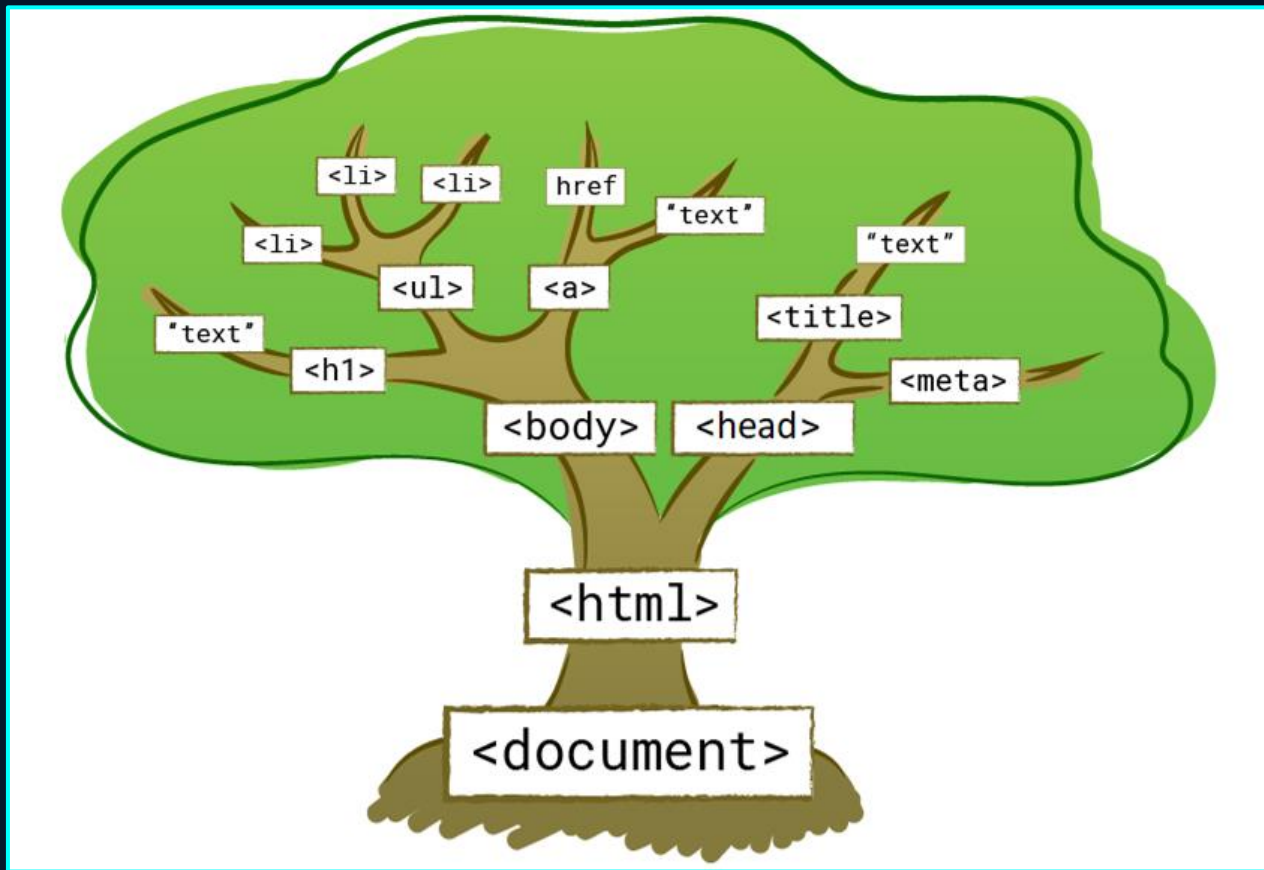
```
npm start
```

# Webpack – Module Bundler



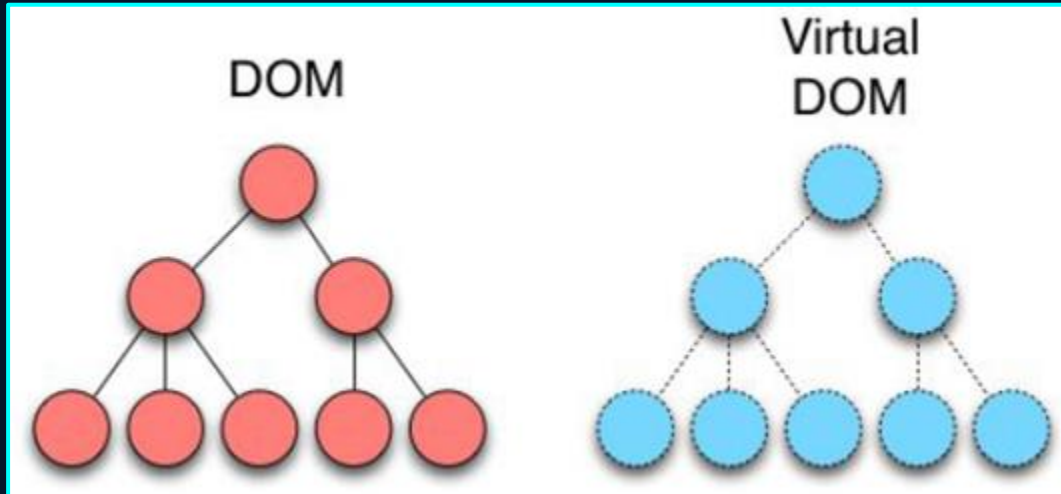
- 🔥 **Webpack** es un module bundler o empaquetador de módulos, que recoge todos los módulos que tenga nuestra app y los transforma en assets que puede entender el navegador.
- 💧 Al utilizar create-react-app, webpack está incluido por defecto, generando estos *bundles* o *paquetes* automáticamente.

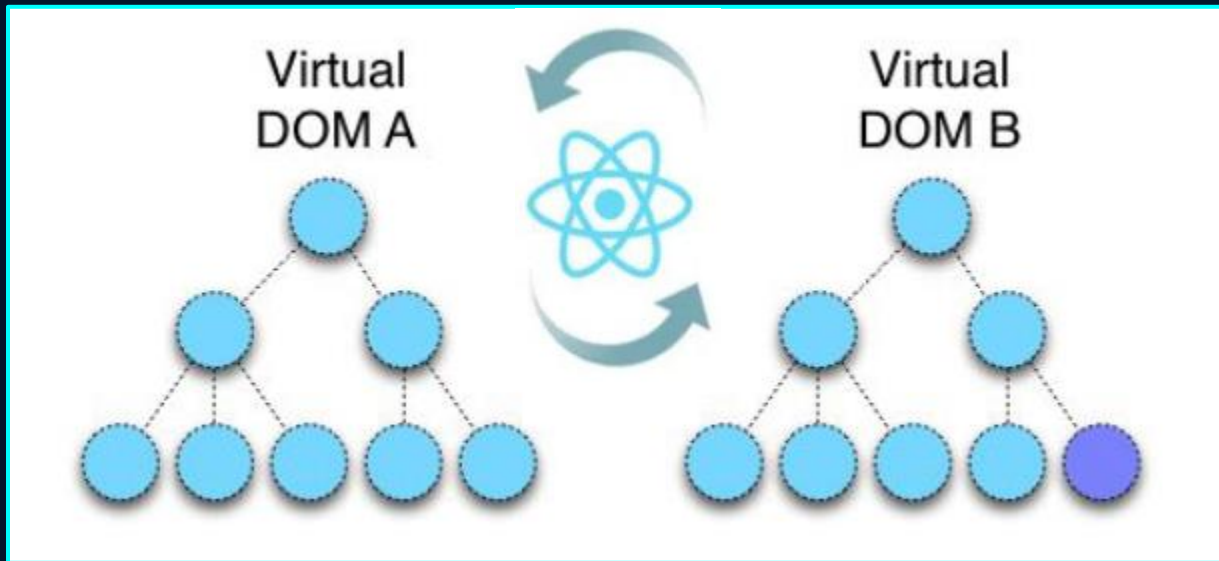
# ¿Recuerdan el DOM?



# Virtual DOM

El Virtual DOM es una copia de nuestro árbol DOM guardada en memoria. Esto es posible porque React representa *virtualmente* nuestra UI y la mantiene sincronizada con el DOM real.





Una vez que React dispone de esta versión del árbol, se encarga de estar pendiente de los cambios que se generan sobre él. Siendo capaz de ver las diferencias entre un estado A y un estado B cuando nosotros actualizamos información. Así reacciona (*react*) a los mismos, actualizando solo lo necesario.

¡¡VAYAMOS AL PROYECTO A  
EXPLORAR LA ARQUITECTURA  
BÁSICA DE REACT!! 🦵 🦵 🦵

# JSX

The background of the slide is a blurred image of a person with long dark hair, seen from the side, wearing large white headphones. They are sitting at a desk. In front of them is a laptop and a larger monitor. The monitor displays a video call with two participants: a woman on the left and a man on the right. The man is wearing a suit and glasses. There are some plants visible in the background, including a large leafy plant on the right side of the desk.

# ¿Qué es JSX?



💧 JSX es una extensión de *sintaxis* de Javascript que se parece a HTML, y nos permite crear los elementos del Virtual DOM que serán renderizados finalmente en el DOM.

💧 Al ser Javascript, podemos añadir lógica dentro del DOM de manera mucho más sencilla.

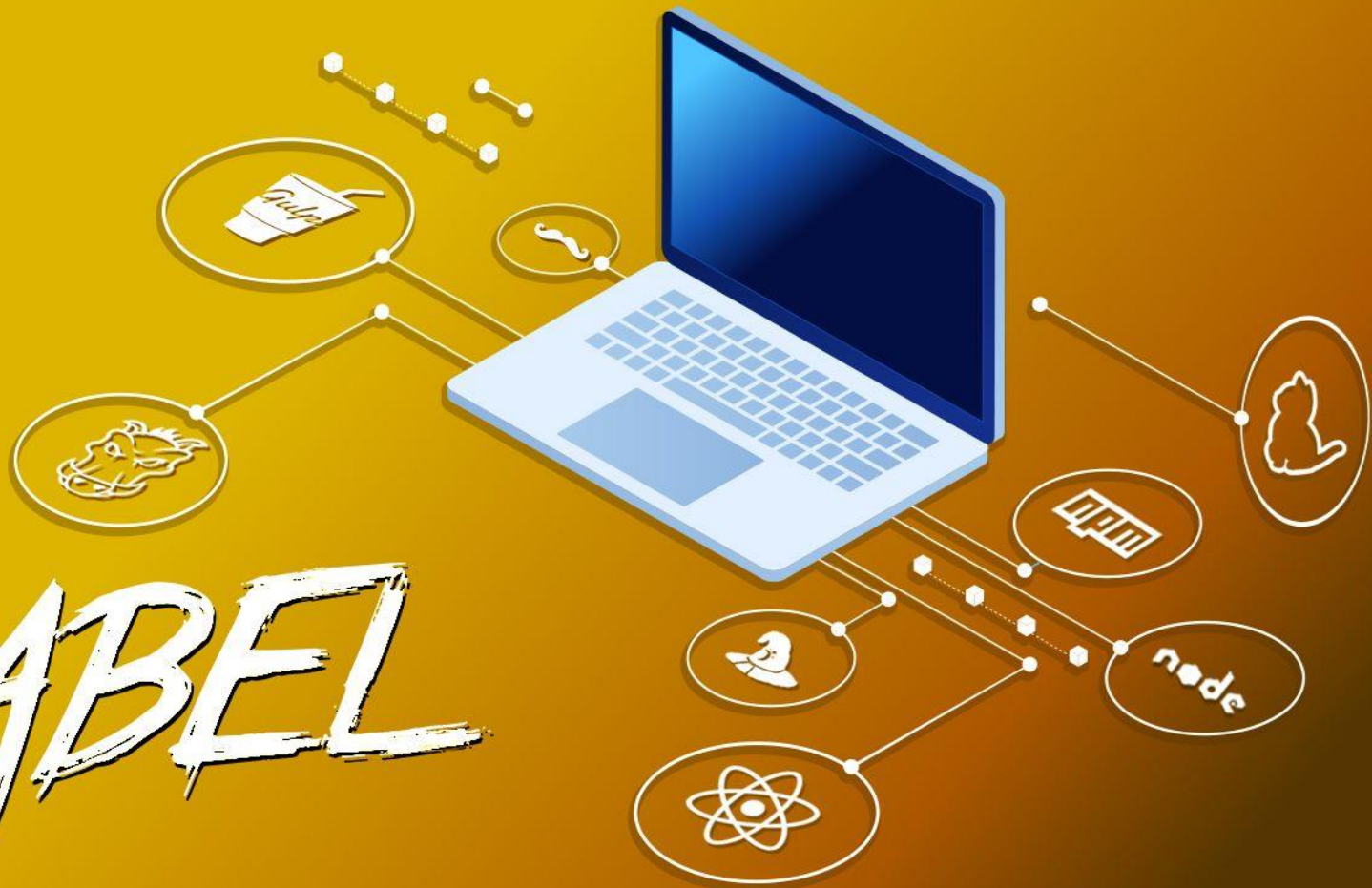


# ¿Qué diferencias tiene con HTML?

- ✓ Todos los elementos del componente deben estar dentro de *un solo padre*.
- ✓ Por cada apertura debe haber un *cierre*.
- ✓ Si el elemento no tiene hijos, debe idealmente ser *auto-cerrado*.
- ✓ Class es palabra reservada, en su lugar usar *className*.
- ✓ Usa la convención de nomenclatura *camelCase* en vez de nombres de atributos HTML.
- ✓ Para añadir sintaxis tradicional de Javascript dentro de JSX, es necesario usar *llaves {}*.

¡¡Veámos cómo funciona JSX en el fondo!!

# BABEL





🔥 **Babel** es un compilador JavaScript, o mejor dicho, un **transpilador**, que reescribe tu código JavaScript para que sea compatible con versiones anteriores de JavaScript.

🔥 Al utilizar create-react-app, babel es configurado automáticamente.

# ¿Qué hay detrás de JSX?

// Ejemplo 1

```
<p id="1" class="Numen">Hola Mundo</p>
```



// Ejemplo 2

```
let cadena = "Hola Mundo"
```

```
<p id="1" class="Numen">{cadena}</p>
```

// Ejemplo 3

```
<p id="1" class="Numen">Hola Mundo</p>
```

```
<p id="2" class="Numen">Hola Adios</p>
```

// Ejemplo 4

```
<>
```

```
<p id="1" class="Numen">Hola Mundo</p>
```

```
<p id="2" class="Numen">Hola Adios</p>
```

```
</>
```

[Link a Babel](#)

🤔 ¿Qué sucede si creo un elemento con una apertura < >, pero sin clausura </>?

```
<img src="" alt="">
```



🤔 ¿Hay forma de mejorarlo?

```
<img src="" alt=""> </img>
```



🤔 ¿Hay alguna otra alternativa que ofrezca React?

```
<img src="" alt="" />
```



🤔 ¿Puedo utilizar `class` para identificar qué clase de estilos corresponde a mi elemento?

```
<h1 class="titulo-grande">Título</h1>
```



⚠️ `Class` es una palabra reservada en ES6. Por lo tanto, no podemos utilizarla en JSX ya que este será luego compilado a Javascript. En su lugar, debemos utilizar `className`:

```
<h1 className="titulo-grande">Título</h1>
```





Si JSX es una extensión de Javascript, ¿puedo añadir Javascript dentro de mis etiquetas? Absolutamente, pero debo recordar que debo envolver dicho código {entre llaves}


```
function App() {  
  
  const texto = "Esto es un texto"  
  
  return (  
    <>  
      <h1>{texto}</h1>  
    </>  
  )  
}  
  
export default App
```

Puedo  
almacenar algo  
en una variable

Luego puedo  
utilizarlo dentro  
de una etiqueta

```
function App() {  
  
  const textos = {  
    textoUno: "Esto es un texto",  
    textoDos: "Esto es un texto",  
    textoTres: "Esto es un texto",  
  }  
  
  return (  
    <>  
      <h1>{textos.textoUno}</h1>  
      <h2>{textos.textoDos}</h2>  
      <h3>{textos.textoTres}</h3>  
    </>  
  )  
}
```

```
export default App
```

 Cualquier cosa que exista en Javascript puede ser utilizada dentro de JSX. Por ejemplo: objetos, funciones, arrays y strings.





¿Puedo escribir estilos inline en JSX? Es posible declarar estilos y atribuírselos a la propiedad `style`, pero JS lo recibirá como un objeto. Ante esto, existen dos alternativas:

```
function App() {  
  
  const estilos = {  
    color: "red",  
    margin: "1rem",  
    padding: 5  
  }  
  
  return (  
    <h1 style={estilos}>Estilos en línea</h1>  
  )  
}  
  
export default App
```

1. La primera es armar un *objeto* con los estilos que deseamos incluir y almacenarlo en una variable. Luego, abrir llaves para introducir JS tradicional e incluir el objeto.

```
function App() {  
  
  return (  
    <h1 style={{  
      color: "red",  
      margin: "1rem",  
      padding: 5  
    }}>  
      Estilos en línea  
    </h1>  
  )  
}  
export default App
```

2. La segunda es utilizar `{ { dobles llaves } }`. Las primeras llaves con la función de introducir JS tradicional dentro de JSX. Las otras llaves para declarar el objeto que contiene los estilos.

```
function App() {  
  
  return (  
    <h1 style={{  
      fontSize: "2rem",  
      paddingTop: "1rem",  
      marginBottom: 10  
    }}>  
      Estilos en línea  
    </h1>  
  )  
}  
  
export default App
```

**i** Al declarar los estilos en un objeto, debes utilizar camelCase.

**i** Si utilizas solo números, sin especificar la unidad de medida, el valor por defecto es *px*.

# Renderizando Listas en JSX

```
function App() {  
  
  const lista = [1, 2, 3, 4, 5]  
  const listaMapeada = lista.map(item =>  
    <li>{item}</li>  
  
    return (  
      <ul>  
        {listaMapeada}  
      </ul>  
    )  
  }  
  export default App
```

🤔 ¿Qué alternativas ofrece React para renderizar listas?

💡 Puedes almacenar tu lista en un array, y luego utilizar el método *map* para iterar a través de la lista.

```
function App() {  
  
  const lista = [  
    {id: "1", text: "item1"},  
    {id: "2", text: "item2"},  
    {id: "3", text: "item3"}  
  ]  
  
  return (  
    <ul>  
      {lista.map(item => (  
        <li key={item.id}>{item.text}</li>  
      ))}  
    </ul>  
  )  
}  
  
export default App
```

⚠ Es necesario asignarle a cada item de nuestra lista una *key*. Una “key” es un atributo especial *string* que es necesario incluir al crear listas de elementos.

- ✓ Manejar eventos en elementos de React es muy similar a manejar eventos con elementos del DOM, con la salvedad de algunas diferencias:
- Los eventos de React se nombran usando *camelCase*, en vez de minúsculas.
  - Con JSX pasas una *función* como el manejador del evento, en vez de un string.

```
<button onClick={alert}>
```

```
  Alarma
```

```
</button>
```

Versión  
React en JSX

```
<button onClick="alert()">
```

```
  Alarma
```

```
</button>
```

Versión Clásica  
en HTML

ACADEMY  
by NUMEN