



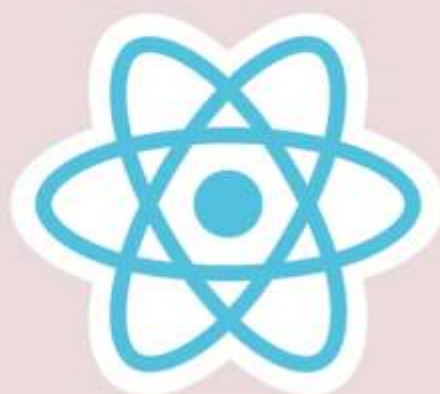
# 5\_1.1

www.opentowork.es

# ReactJS

## Introducción

ReactJS, también conocido simplemente como React, es una biblioteca de JavaScript para construir interfaces de usuario. Fue desarrollada por Facebook y es mantenida por Facebook y una comunidad de desarrolladores individuales y empresas. React se utiliza principalmente para crear aplicaciones web de una sola página, permitiendo a los desarrolladores crear interfaces de usuario complejas a partir de pequeños y aislados pedazos de código llamados "componentes".



#01/24@ mihifidem



# Tema 1: Introducción a ReactJS

## 1. Introducción a ReactJS

- 1.1. ¿Qué es ReactJS y por qué usarlo?
- 1.2. Beneficios y características principales de ReactJS.
- 1.3. Breve comparación con otros frameworks o bibliotecas.
- 1.4. Resumen

## 2. Primeros pasos

- 2.1. Directamente en HTML
- 2.2. Configuración de un entorno de desarrollo
- 2.3. Ejecute la aplicación React
- 2.4. Modificando la aplicación React
- 2.5. Actualizar React

## 3. React ES6

- 3.1. ¿Qué es ES6?
- 3.2. Clases de React ES6
- 3.3. Funciones de flecha de React ES6
- 3.4. React ES6 Variables
- 3.5. Métodos de matriz React ES6
- 3.6. React ES6 Desestructuración
- 3.7. Operador de propagación React ES6
- 3.8. Módulos React ES6
- 3.9. Operador ternario de React ES6

## 4. React render HTML

- 4.1. Function createRoot
- 4.2. El método de render
- 4.3. El código HTML
- 4.4. El nodo root

## 5. React JSX

- 5.1. ¿Qué es JSX?
- 5.2. Codificación JSX
- 5.3. Expresiones en JSX
- 5.4. Insertar un bloque grande de HTML
- 5.5. Un elemento de nivel superior
- 5.6. Attribute class = className
- 5.7. Condiciones - sentencias if

## 6. Componentes en ReactJS

- 6.1. Concepto de componentes y su importancia en ReactJS
- 6.2. Componentes funcionales vs. componentes de clase.
- 6.3. Creación de componentes funcionales y de clase.
- 6.4. Renderización de elementos JSX en los componentes.
- 6.5. RESUMEN



## 1. Introducción a ReactJS

### 1.1. ¿Qué es ReactJS y por qué usarlo?

ReactJS es una **biblioteca de JavaScript de código abierto** utilizada para construir interfaces de usuario interactivas y reactivas. Se centra en la creación de componentes reutilizables que encapsulan el comportamiento y la presentación de partes específicas de la interfaz de usuario.

Existen varias razones por las cuales ReactJS es ampliamente utilizado y preferido por los desarrolladores:

**Eficiencia y rendimiento:** ReactJS utiliza un modelo de renderizado eficiente llamado "Virtual DOM". En lugar de actualizar directamente el DOM cada vez que se produce un cambio, ReactJS compara el Virtual DOM con el DOM real y solo realiza las actualizaciones necesarias, lo que resulta en un rendimiento más rápido y una mejor eficiencia.

**Componentes reutilizables:** ReactJS fomenta la creación de componentes reutilizables y modulares. Los componentes encapsulan tanto la estructura HTML como la lógica relacionada, lo que facilita su reutilización en diferentes partes de la aplicación y en proyectos futuros. Esto mejora la legibilidad, el mantenimiento y la escalabilidad del código.

**Unidireccionalidad de los datos:** ReactJS sigue el flujo de datos unidireccional, lo que significa que los datos fluyen en una dirección clara desde el componente padre hacia los componentes hijos. Esta arquitectura simplifica el seguimiento y la depuración de los cambios de datos, lo que resulta en un código más predecible y menos propenso a errores.

**Comunidad activa y ecosistema robusto:** ReactJS cuenta con una comunidad de desarrolladores muy activa y un amplio ecosistema de bibliotecas y herramientas de soporte. Esto facilita la obtención de soluciones, la colaboración y el aprendizaje continuo.

**React Native:** ReactJS sirve como la base de React Native, un marco de desarrollo para crear aplicaciones móviles nativas. Al compartir gran parte del código entre aplicaciones web y móviles, ReactJS permite el desarrollo rápido y eficiente de aplicaciones multiplataforma.

En resumen, ReactJS es una opción popular para el desarrollo de interfaces de usuario debido a su eficiencia, rendimiento, modularidad y unidireccionalidad de los datos. Al utilizar ReactJS, los desarrolladores pueden construir aplicaciones web rápidas, escalables y fáciles de mantener.



## 1.2. Beneficios y características principales de ReactJS.

ReactJS ofrece una serie de beneficios y características clave que lo convierten en una elección popular para el desarrollo de aplicaciones web. Estos son algunos de los beneficios más destacados de ReactJS:

**Virtual DOM:** ReactJS utiliza un Virtual DOM (DOM virtual) para realizar actualizaciones eficientes en la interfaz de usuario. En lugar de manipular directamente el DOM del navegador, ReactJS crea una representación virtual del DOM y solo actualiza los elementos que han cambiado, lo que mejora el rendimiento y la eficiencia.

**Componentes reutilizables:** ReactJS fomenta el desarrollo de componentes reutilizables, lo que permite crear piezas de interfaz de usuario autónomas y modulares. Los componentes se pueden combinar para construir interfaces complejas, lo que facilita la reutilización de código y la creación de aplicaciones escalables.

**Unidireccionalidad de los datos:** ReactJS sigue el flujo de datos unidireccional, lo que significa que los datos fluyen en una sola dirección, desde el componente padre hacia los componentes hijos. Esto hace que el seguimiento de los cambios de datos sea más fácil y predecible, evitando problemas de sincronización y mejorando la depuración del código.

**JSX:** ReactJS utiliza JSX, una extensión de sintaxis que permite escribir código JavaScript y HTML en conjunto. JSX simplifica la creación de componentes y la composición de elementos de interfaz de usuario, proporcionando una sintaxis más intuitiva y legible.

**React Native:** ReactJS sirve como base para React Native, un marco de desarrollo para crear aplicaciones móviles nativas. Esto significa que gran parte del código de ReactJS se puede reutilizar para construir aplicaciones móviles, lo que ahorra tiempo y esfuerzo en el desarrollo de aplicaciones multiplataforma.

**Comunidad activa y ecosistema robusto:** ReactJS cuenta con una comunidad de desarrolladores muy activa y un amplio ecosistema de bibliotecas y herramientas de soporte. Hay una gran cantidad de recursos, documentación y ejemplos disponibles, lo que facilita la resolución de problemas, la colaboración y el aprendizaje continuo.

**Integración sencilla:** ReactJS se puede integrar fácilmente con otras bibliotecas y marcos de JavaScript, lo que brinda flexibilidad y opciones para adaptarse a las necesidades específicas del proyecto.

En general, ReactJS ofrece una combinación única de rendimiento, modularidad, reutilización de componentes y una comunidad activa que lo convierten en una opción poderosa y popular para el desarrollo de aplicaciones web.



## 1.3. Breve comparación con otros frameworks o bibliotecas.

Aquí tienes una breve comparación de ReactJS con otros frameworks o bibliotecas populares:

**Angular:** ReactJS y Angular son dos frameworks populares para el desarrollo de aplicaciones web. Mientras que ReactJS se centra en la construcción de interfaces de usuario y utiliza un Virtual DOM para el renderizado eficiente, Angular es un framework completo que ofrece un enfoque más amplio para el desarrollo web, incluyendo enrutamiento, gestión del estado y más. ReactJS tiende a ser más flexible y se integra fácilmente con otras bibliotecas, mientras que Angular proporciona una estructura más rígida y un conjunto de características más completo.

**Vue.js:** Vue.js es otra biblioteca popular para la construcción de interfaces de usuario. Al igual que ReactJS, Vue.js se centra en componentes reutilizables y utiliza un enfoque basado en Virtual DOM. Vue.js es conocido por su curva de aprendizaje más suave y su sintaxis simple. ReactJS tiene una comunidad más grande y una adopción más amplia, pero Vue.js ofrece una experiencia de desarrollo rápida y fácil de entender para proyectos más pequeños o para desarrolladores que recién comienzan.

**Ember.js:** Ember.js es un framework de JavaScript completo para la construcción de aplicaciones web. A diferencia de ReactJS, Ember.js ofrece una estructura más completa y estándares definidos para el desarrollo de aplicaciones, incluyendo manejo de rutas, administración de datos y pruebas. Ember.js es ideal para aplicaciones web grandes y complejas, mientras que ReactJS es más adecuado para proyectos más pequeños o donde se requiere una mayor flexibilidad.

**jQuery:** jQuery es una biblioteca de JavaScript que se centra en la manipulación del DOM y la simplificación de tareas comunes de JavaScript en el navegador. A diferencia de ReactJS, jQuery no se enfoca en la construcción de componentes reutilizables ni en el enfoque basado en Virtual DOM. Es más adecuado para proyectos más pequeños que requieren manipulación directa del DOM y operaciones sencillas.

En general, ReactJS se destaca por su enfoque en la construcción de componentes reutilizables, su eficiente renderizado basado en Virtual DOM y su amplia comunidad de desarrolladores. La elección entre ReactJS y otros frameworks o bibliotecas dependerá de las necesidades del proyecto, la escala del desarrollo y las preferencias del equipo de desarrollo.

## 1.4. RESUMEN

### ¿Qué es React?

- React, a veces denominado framework de JavaScript frontend, es una biblioteca de JavaScript creada por Facebook.
- React es una herramienta para construir componentes de interfaz de usuario.
- React es una biblioteca de JavaScript para crear interfaces de usuario.
- React se utiliza para crear aplicaciones de una sola página.
- React nos permite crear componentes de interfaz de usuario reutilizables.

### ¿Cómo funciona React?

- React crea un DOM VIRTUAL en la memoria.
- En lugar de manipular el DOM del navegador directamente, React crea un DOM virtual en la memoria, donde realiza toda la manipulación necesaria antes de realizar los cambios en el DOM del navegador.
- ¡React solo cambia lo que necesita ser cambiado!
- React descubre qué cambios se han realizado y cambia solo lo que debe cambiarse.

### Historia de React.JS

- La versión actual de React.JS es V18.0.0 (abril de 2022).
- El lanzamiento inicial al público (V0.3.0) fue en julio de 2013.
- React.JS se utilizó por primera vez en 2011 para la función Newsfeed de Facebook.
- El ingeniero de software de Facebook, Jordan Walke, lo creó.
- La versión actual de **create-react-app** es v5.0.1 (abril de 2022).
- **create-react-app** incluye herramientas integradas como webpack, Babel y ESLint.

## 2. Primeros pasos

- Para usar React en producción, necesita npm, que se incluye con [Node.js](#).
- Para obtener una descripción general de lo que es React, puede escribir código React directamente enHTML.
- Pero para usar React en producción, necesita npm y [Node.js](#) instalados.

### 2.1. Directamente en HTML

La forma más rápida de comenzar a aprender React es escribir React directamente en sus archivos HTML.

Comience por incluir tres scripts, los dos primeros nos permiten escribir código React en nuestros JavaScripts, y el tercero, Babel, nos permite escribir sintaxis JSX y ES6 en navegadores más antiguos.

#### Ejemplo 1

Incluya tres CDN en su archivo HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  </head>
  <body>

    <div id="mydiv"></div>

    <script type="text/babel">
      function Hello() {
        return <h1>Hello World!</h1>;
      }

      const container = document.getElementById('mydiv');
      const root = ReactDOM.createRoot(container);
      root.render(<Hello />)
    </script>

  </body>
</html>
```

Esta forma de usar React puede estar bien para fines de prueba, pero para la producción deberá configurar un entorno React .

## 2.2. Configuración de un entorno de react

Si tiene instalados npx y Node.js, puede crear una aplicación React usando create-react-app.

### ¿npx es lo mismo que npm?

No, npx no es lo mismo que npm, aunque ambos son herramientas relacionadas con Node.js y el ecosistema de JavaScript. Aquí tienes una explicación de cada uno:

**npm (Node Package Manager):** npm es el gestor de paquetes predeterminado para Node.js. Se utiliza para instalar, administrar y actualizar paquetes y dependencias de JavaScript. Con npm, puedes descargar y utilizar bibliotecas de código de terceros en tu proyecto, administrar las versiones de las dependencias y ejecutar scripts personalizados definidos en el archivo package.json. npm se instala junto con Node.js y se ejecuta a través de la línea de comandos con comandos como npm install, npm start, etc.

**npx (Node Package Runner):** npx es una herramienta que se incluye en npm desde la versión 5.2.0. Su objetivo principal es ejecutar paquetes de Node.js de manera más conveniente. A diferencia de npm install -g, que instala paquetes de forma global en tu sistema, npx se utiliza para ejecutar paquetes sin necesidad de instalarlos globalmente. Puedes usar npx para ejecutar comandos de línea de comandos de paquetes específicos sin tener que instalarlos previamente. Por ejemplo, puedes ejecutar npx create-react-app my-app para crear una nueva aplicación de React sin necesidad de tener la herramienta create-react-app instalada globalmente.

En resumen, npm se utiliza principalmente para instalar y administrar paquetes y dependencias en tu proyecto, mientras que npx se utiliza para ejecutar paquetes de Node.js de forma temporal y conveniente sin necesidad de instalación previa.

Si anteriormente realizó una instalación global **create-react-app**, se recomienda que desinstale el paquete para asegurarse de que npx siempre use la última versión de **create-react-app**.

Para desinstalar, ejecute este comando: **npm uninstall -g create-react-**

**app**. Ejecute este comando para crear una aplicación React llamada **my-react-app**:

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



# Tema 1: Introducción a ReactJS



## 2.3. Ejecute la aplicación React

¡Ahora está listo para ejecutar su primera aplicación React

real !Ejecute este comando para moverse al directorio **my-**

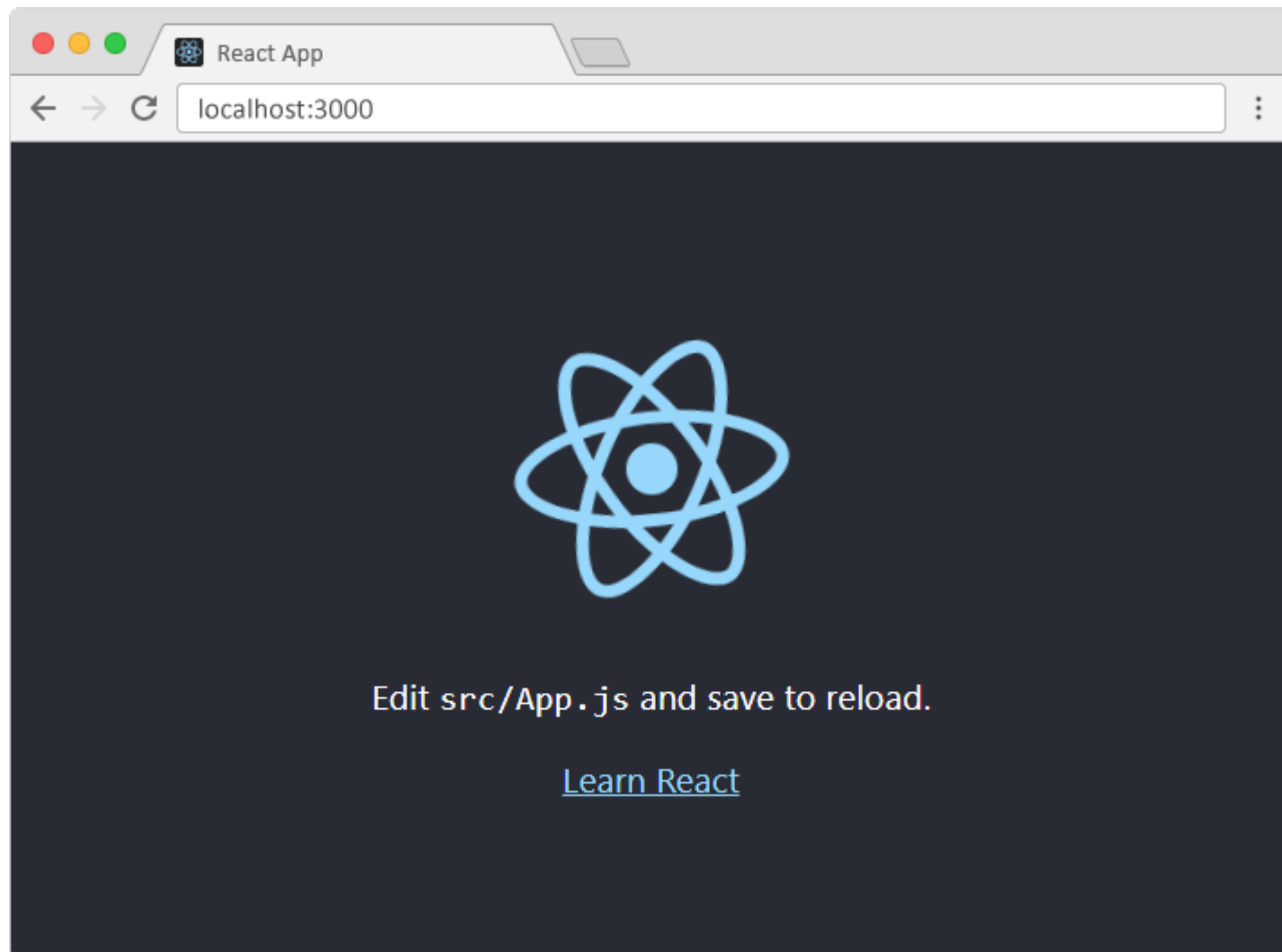
**react-app:**

```
cd my-react-app
```

```
npm start
```

Aparecerá una nueva ventana del navegador con su aplicación React recién creada! Si no, abra su navegador y escriba localhost:3000 en la barra de direcciones.

El resultado:



# Tema 1: Introducción a ReactJS



## 2.4. Modificar la aplicación React

Hasta aquí todo bien, pero ¿cómo cambio el contenido?

Busque en el directorio **my-react-app** y encontrará una carpeta **src**. Dentro de la carpeta src hay un archivo llamado App.js, ábrelo y se verá así:

/miReactApp/src/App.js:

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Intente cambiar el contenido HTML y guarde el archivo.

Tenga en cuenta que los cambios son visibles inmediatamente después de guardar el archivo, ¡no es necesario que vuelva a cargar el navegador!



## Tema 1: Introducción a ReactJS

Reemplace todo el contenido dentro de `<div className="App">` con un elemento

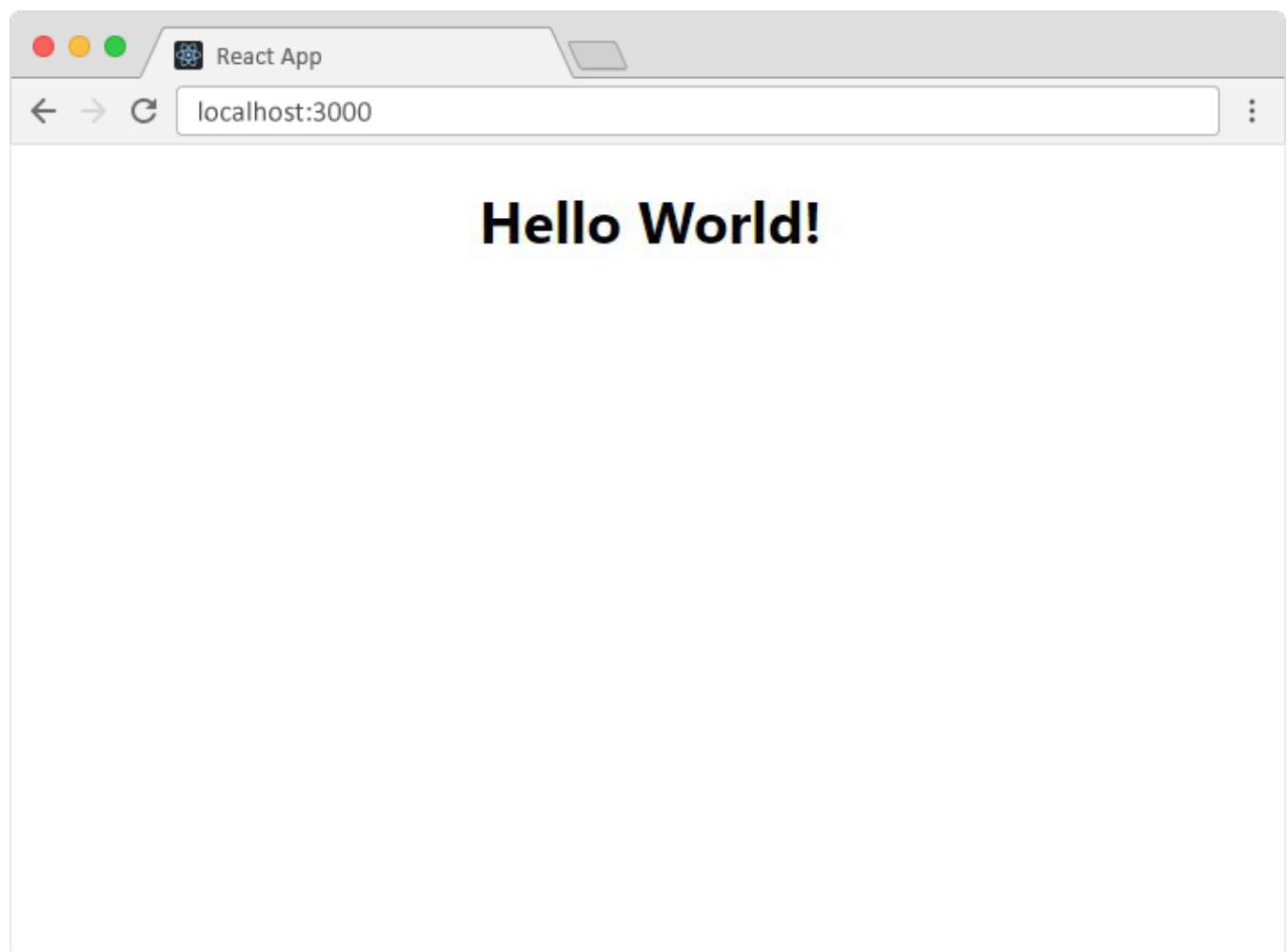
`<h1>`. Vea los cambios en el navegador cuando haga clic en Guardar.

```
function App() {  
  return (  
    <div className="App">  
      <h1>Hello World!</h1>  
    </div>  
  );  
}
```

`export default App;`

Tenga en cuenta que hemos eliminado las importaciones que no necesitamos (logo.svg y App.css).

El resultado:



# Tema 1: Introducción a ReactJS



Ahora tiene un entorno React en su computadora y está listo para aprender más sobre React.

Procederemos a desmontar la carpeta src para que solo contenga un archivo: index.js. También debe eliminarlas líneas de código innecesarias dentro del index.js archivo para que se vean como el ejemplo en la herramienta.

## Ejemplo 3

index.js:

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

const myFirstElement = <h1>Hello React!</h1>

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myFirstElement);

/*
You are now watching
the React file 'index.js'
through our 'Show React' tool.
*/
```



## 2.5. Actualizar React

### Actualizar a React 18

Actualizar una aplicación React existente a la versión 18 solo requiere dos pasos.

Si ya está utilizando la última versión que **create-react-app** utiliza React versión 18, puede omitir esta sección.

### Paso 1: Instalar React 18

Para instalar la última versión, desde la carpeta de su proyecto, ejecute lo siguiente desde la terminal:

```
npm i react@latest react-dom@latest
```

### Paso 2: use la nueva API raíz

Para aprovechar las funciones simultáneas de React 18, deberá usar la nueva API raíz para la representación del cliente.

```
// Before
import ReactDOM from 'react-dom';
ReactDOM.render(<App />, document.getElementById('root'));
```

```
// After
import ReactDOM from 'react-dom/client';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Su aplicación funcionará sin usar la nueva API raíz. Si continúa usando, ReactDOM.render su aplicación se comportará como React 17.

## 3. React ES6

### 3.1. ¿Qué es ES6?

ES6 significa ECMAScript 6.

ECMAScript se creó para estandarizar JavaScript, y ES6 es la sexta versión de ECMAScript, se publicó en 2015 y también se conoce como ECMAScript 2015.

### ¿Por qué debo aprender ES6?

React usa ES6, y debería estar familiarizado con algunas de las nuevas características como:

- Clases
- Funciones de flecha
- Variables (let, const, var)
- Métodos de Array como .map()
- Desestructuración
- Módulos
- Operador Ternario
- Operador de propagación

### ¿ React ES6 es lo mismo ReactJS?

No, React ES6 no es lo mismo que ReactJS, aunque están relacionados. A continuación, te explico brevemente cada uno:

**React ES6** (también conocido como React con ECMAScript 6): React es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y reactivas. React ES6 se refiere al uso de React en combinación con las características y sintaxis introducidas en la especificación ECMAScript 6 (también conocida como ES6 o ES2015). ECMAScript 6 es una versión del estándar JavaScript que se lanzó en 2015 e introdujo nuevas características y mejoras en el lenguaje, como la sintaxis de clases, las funciones de flecha, los módulos, entre otros. React se puede utilizar con cualquier versión de JavaScript, pero aprovechar las características de ES6 puede hacer que el código sea más conciso y legible.

**ReactJS:** ReactJS es el término comúnmente utilizado para referirse a React, la biblioteca de JavaScript mencionada anteriormente. Es una biblioteca de código abierto ampliamente utilizada para construir interfaces de usuario en aplicaciones web. ReactJS permite crear componentes reutilizables que gestionan su propio estado y se actualizan eficientemente cuando cambia el estado de la aplicación. Con ReactJS, puedes construir aplicaciones de una sola página (Single-Page Applications) o aplicaciones más complejas utilizando enfoques como el enrutamiento y la gestión del estado global.

En resumen, React ES6 se refiere al uso de React con las características y sintaxis de ECMAScript 6, mientras que ReactJS es la biblioteca de JavaScript utilizada para construir interfaces de usuario reactivas en aplicaciones web. ReactJS se puede utilizar tanto con la sintaxis de ES6 como con versiones anteriores de JavaScript, pero aprovechar las características de ES6 puede hacer que el código sea más moderno y expresivo.

## 3.2. Clases de React ES6

### Clases

ES6 introdujo clases.

Una clase es un tipo de función, pero en lugar de usar la palabra clave **function** para iniciarla, usamos la palabra clave **class** y las propiedades se asignan dentro de un método **constructor()**.

### Ejemplo

Un constructor de clase simple:

```
class Car {  
  constructor(name) {  
    this.brand = name;  
  }  
}
```

Observe el caso del nombre de la clase. Hemos comenzado el nombre, "Car", con un carácter en mayúscula. Esta es una convención de nomenclatura estándar para las clases.

Ahora puedes crear objetos usando la clase Car:

### Ejemplo

Cree un objeto llamado "mycar" basado en la clase Car:

```
<!DOCTYPE html>  
<html>  
  
<body>  
  
<script>  
class Car {  
  constructor(name) {  
    this.brand = name;  
  }  
}  
const mycar = new Car("Ford");  
  
document.write(mycar.brand);  
</script>  
  
</body>  
</html>
```

Nota: La función constructora se llama automáticamente cuando se inicializa el objeto.

# Tema 1: Introducción a ReactJS



## Método en Clases

Puede agregar sus propios métodos en una clase:

### Ejemplo

Cree un método llamado "present":

```
class Car {  
  constructor(name) {  
    this.brand = name;  
  }  
  
  present() {  
    return 'I have a ' + this.brand;  
  }  
}  
  
const mycar = new Car("Ford");  
mycar.present();
```

Como puede ver en el ejemplo anterior, llama al método haciendo referencia al nombre del método del objetoseguido de paréntesis (los parámetros irían dentro de los paréntesis).





# Tema 1: Introducción a ReactJS



## Herencia de clase

Para crear una herencia de clase, utilice la palabra clave **extends**.

Una clase creada con una herencia de clase hereda todos los métodos de otra clase:

### Ejemplo

Cree una clase llamada "**Model**" que heredará los métodos de la clase "**Car**":

```
class Car {  
  constructor(name) {  
    this.brand = name;  
  }  
  
  present() {  
    return 'I have a ' + this.brand;  
  }  
}  
  
class Model extends Car {  
  constructor(name, mod) {  
    super(name);  
    this.model = mod;  
  }  
  show() {  
    return this.present() + ', it is a ' + this.model  
  }  
}  
  
const mycar = new Model("Ford", "Mustang");  
mycar.show();
```

El método **super()** se refiere a la clase padre.

Al llamar al método **super()** en el método constructor, llamamos al método constructor del padre y obtenemos acceso a las propiedades y métodos del padre.



## Tema 1: Introducción a ReactJS

### 3.3. Funciones de flecha de React ES6

#### Funciones de flecha

Las funciones de flecha nos permiten escribir una sintaxis de función más corta:

#### Ejemplo

Antes:

```
hello = function() {  
  return "Hello World!";  
}
```

#### Ejemplo

Con función de flecha:

```
hello = () => {  
  return "Hello World!";  
}
```

¡Se hace más corto! Si la función tiene solo una declaración y la declaración devuelve un valor, puede eliminarlos corchetes y la palabra clave **return**:

#### Ejemplo

Las funciones de flecha devuelven valor por defecto:

```
hello = () => "Hello World!";
```

Si tiene parámetros, los pasa dentro de los paréntesis:

#### Ejemplo

Función de flecha con parámetros:

```
hello = (val) => "Hello " + val;
```

## Tema 1: Introducción a ReactJS

De hecho, si solo tiene un parámetro, también puede omitir los paréntesis:

### Ejemplo

Función de flecha sin paréntesis:

```
hello = val => "Hello " + val;
```

### ¿Qué pasa this?

El manejo de **this** también es diferente en las funciones de flecha en comparación con las funciones regulares. En resumen, con las funciones de flecha no hay vinculación de **this**.

En las funciones regulares la palabra clave **this** representaba el objeto que llamaba a la función, que podía ser la ventana, el documento, un botón o lo que fuera.

Con las funciones de flecha, la palabra clave **this** siempre representa el objeto que definió la función de flecha. Echemos un vistazo a dos ejemplos para entender la diferencia.

Ambos ejemplos llaman a un método dos veces, primero cuando se carga la página y una vez más cuando el usuario hace clic en un botón.

El primer ejemplo usa una función regular y el segundo ejemplo usa una función de flecha.

El resultado muestra que el primer ejemplo devuelve dos objetos diferentes (ventana y botón) y el segundo ejemplo devuelve el objeto de encabezado dos veces.

# Tema 1: Introducción a ReactJS



## Ejemplo

Con una función regular, **this** representa el objeto que llamó a la función:

```
class Header {  
  constructor() {  
    this.color = "Red";  
  }  
  
  //Regular function:  
  changeColor = function() {  
    document.getElementById("demo").innerHTML += this;  
  }  
}  
  
const myheader = new Header();  
  
//The window object calls the function:  
window.addEventListener("load", myheader.changeColor);  
  
//A button object calls the function:  
document.getElementById("btn").addEventListener("click", myheader.changeColor);
```

# Tema 1: Introducción a ReactJS



## Ejemplo

Con una función de flecha, **this** representa el objeto de encabezado sin importar quién llamó a la función:

```
class Header {
  constructor() {
    this.color = "Red";
  }

  //Arrow function:
  changeColor = () => {
    document.getElementById("demo").innerHTML += this;
  }
}

const myheader = new Header();

//The window object calls the function:
window.addEventListener("load", myheader.changeColor);

//A button object calls the function:
document.getElementById("btn").addEventListener("click", myheader.changeColor);
```

Recuerde estas diferencias cuando trabaje con funciones. A veces, el comportamiento de las funciones regulares es lo que desea, si no, use funciones de flecha.



## 3.4. React ES6 Variables

### Variables

Antes de ES6, solo había una forma de definir sus variables: con la palabra clave **var**. Si no los definió, se asignarían al objeto global. A menos que estuviera en modo estricto, obtendría un error si sus variables no estuvieran definidas.

Ahora, con ES6, hay tres formas de definir sus variables: **var**, **let** y **const**.

### Ejemplo

#### **var**

```
var x = 5.6;
```

Si usa **var** fuera de una función, pertenece al alcance global. Si usa **var** dentro de una función, pertenece a esa función.

Si usa **var** dentro de un bloque, es decir, un bucle for, la variable todavía está disponible fuera de ese bloque.

**var** tiene un alcance de función, no un alcance de bloque.

### Ejemplo

#### **let**

```
let x = 5.6;
```

**let** es la versión de ámbito de bloque de **var** y se limita al bloque (o expresión) donde se define.

Si usa **let** el interior de un bloque, es decir, un bucle for, la variable solo está disponible dentro de ese bucle.

**Let** tiene un alcance de bloque.

# Tema 1: Introducción a ReactJS

## Ejemplo

### const

```
const x = 5.6;
```

**const** es una variable que una vez creada, su valor nunca puede cambiar.

**const** tiene un alcance de bloque .

La palabra clave **const** es un poco engañosa.

No define un valor constante. Define una referencia constante a un

valor. Por eso NO puedes:

- Reasignar un valor constante
- Reasignar una matriz constante
- Reasignar un objeto

constante Pero puedes:

- Cambiar los elementos de la matriz constante
- Cambiar las propiedades del objeto constante

# Tema 1: Introducción a ReactJS

## 3.5. Métodos de matriz React ES6

### Métodos de matriz

Hay muchos métodos de matriz de JavaScript.

Uno de los más útiles en React es el método `.map()` de matriz.

El método `.map()` le permite ejecutar una función en cada elemento de la matriz, devolviendo una nueva matriz como resultado.

En React, `map()` se puede usar para generar listas.

### Ejemplo

Genere una lista de elementos de una matriz:

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

const myArray = ['apple', 'banana', 'orange'];

const myList = myArray.map((item) => <p>{item}</p>)

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(myList);
```





## 3.6. React ES6 Desestructuración

### desestructuración

Para ver la desestructuración, haremos un sándwich. ¿Saca todo de la nevera para hacer su bocadillo? No, solo saca los artículos que le gustaría usar en su sándwich.

La desestructuración es exactamente lo mismo. Es posible que tengamos una matriz u objeto con el que estemos trabajando, pero solo necesitamos algunos de los elementos contenidos en estos.

La desestructuración facilita extraer solo lo que se necesita.

### Destrucción de array

Aquí está la forma antigua de asignar elementos de matriz a una variable:

#### Ejemplo

Antes:

```
const vehicles = ['mustang', 'f-150', 'expedition'];  
// old way  
const car = vehicles[0];  
const truck = vehicles[1];  
const suv = vehicles[2];
```

Esta es la nueva forma de asignar elementos de matriz a una variable:

#### Ejemplo

Con desestructuración:

```
const vehicles = ['mustang', 'f-150', 'expedition'];  
const [car, truck, suv] = vehicles;
```

Al desestructurar arreglos, el orden en que se declaran las variables es importante.

Si solo queremos el automóvil y el todoterreno, simplemente podemos omitir el camión pero mantener la coma:

```
const vehicles = ['mustang', 'f-150', 'expedition'];  
const [car,, suv] = vehicles;
```

## Tema 1: Introducción a ReactJS

La desestructuración es útil cuando una función devuelve una matriz:

### Ejemplo

```
function calculate(a, b) {  
  const add = a + b;  
  const subtract = a - b;  
  const multiply = a * b;  
  const divide = a / b;  
  
  return [add, subtract, multiply, divide];  
}  
  
const [add, subtract, multiply, divide] = calculate(4, 7);
```

### Destrucción de objetos

Aquí está la forma antigua de usar un objeto dentro de una función:

### Ejemplo

Antes:

```
const vehicleOne = {  
  brand: 'Ford',  
  model: 'Mustang',  
  type: 'car',  
  year: 2021,  
  color: 'red'  
}  
  
myVehicle(vehicleOne);  
  
// old way  
function myVehicle(vehicle) {  
  const message = 'My ' + vehicle.type + ' is a ' + vehicle.color + ' ' + vehicle.brand + ' ' + vehicle.model + '.';  
}
```

## Tema 1: Introducción a ReactJS

Aquí está la nueva forma de usar un objeto dentro de una función:

### Ejemplo

Con desestructuración:

```
const vehicleOne = {  
  brand: 'Ford',  
  model: 'Mustang',  
  type: 'car',  
  year: 2021,  
  color: 'red'  
}  
  
myVehicle(vehicleOne);  
  
function myVehicle({type, color, brand, model}) {  
  const message = 'My ' + type + ' is a ' + color + ' ' + brand + ' ' + model + '.';  
}
```

Tenga en cuenta que las propiedades del objeto no tienen que declararse en un orden específico.

Incluso podemos desestructurar objetos profundamente anidados haciendo referencia al objeto anidado y luego usando dos puntos y llaves para desestructurar nuevamente los elementos necesarios del objeto anidado:

### Ejemplo

```
const vehicleOne = {  
  brand: 'Ford',  
  model: 'Mustang',  
  type: 'car',  
  year: 2021,  
  color: 'red',  
  registration: {  
    city: 'Houston',  
    state: 'Texas',  
    country: 'USA'  
  }  
}  
  
myVehicle(vehicleOne)  
  
function myVehicle({ model, registration: { state } }) {  
  const message = 'My ' + model + ' is registered in ' + state + '.';  
}
```

## 3.7. Operador de propagación React ES6

### Operador de propagación

El operador de extensión de JavaScript ( ...) nos permite copiar rápidamente todo o parte de una matriz u objeto existente en otra matriz u objeto.

#### Ejemplo

```
<!DOCTYPE html>
<html>

<body>

<script>
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];

document.write(numbersCombined);
</script>

</body>
</html>
```

1,2,3,4,5,6

# Tema 1: Introducción a ReactJS



El operador de extensión se usa a menudo en combinación con la desestructuración.

## Ejemplo

Asigne el primer y el segundo elemento de **numbers** a las variables y coloque el rest en una matriz:

```
<!DOCTYPE html>
<html>

<body>

<script>
const numbers = [1, 2, 3, 4, 5, 6];

const [one, two, ...rest] = numbers;

document.write("<p>" + one + "</p>");
document.write("<p>" + two + "</p>");
document.write("<p>" + rest + "</p>");
</script>

</body>
</html>
```

1

2

3,4,5,6



# Tema 1: Introducción a ReactJS



También podemos usar el operador de propagación con objetos:

## Ejemplo

Combina estos dos objetos:

```
<!DOCTYPE html>
<html>

<body>

<script>
const myVehicle = {
  brand: 'Ford',
  model: 'Mustang',
  color: 'red'
}

const updateMyVehicle = {
  type: 'car',
  year: 2021,
  color: 'yellow'
}

const myUpdatedVehicle = {...myVehicle, ...updateMyVehicle}

//Check the result object in the console:
console.log(myUpdatedVehicle);
</script>

<p>Press F12 and see the result object in the console view.</p>

</body>
</html>
```

Observe que las propiedades que no coincidían se combinaron, pero la propiedad que sí coincidió, color, fue sobrescrita por el último objeto que se pasó, updateMyVehicle. El color resultante ahora es amarillo.



## 3.8. Módulos React ES6

### Módulos

Los módulos de JavaScript le permiten dividir su código en archivos

separados. Esto facilita el mantenimiento del código base.

Los módulos ES se basan en las declaraciones import y export.

### Exportar

Puede exportar una función o variable desde cualquier archivo.

Vamos a crear un archivo llamado person.js y llenarlo con las cosas que queremos

exportar. Hay dos tipos de exportaciones: Nombradas y Predeterminadas.

### Exportaciones Named (nombradas)

Puede crear exportaciones con nombre de dos formas. En línea individualmente, o todos a la vez en la parte inferior.

#### Ejemplo

En línea

individualmente:

person.js

```
export const name = "Jesse"
export const age = 40
```

Todo a la vez en la parte

inferior: person.js

```
const name = "Jesse"
const age = 40

export { name, age }
```

# Tema 1: Introducción a ReactJS



## Exportaciones predeterminadas

Vamos a crear otro archivo, llamado message.js, y usarlo para demostrar la exportación predeterminada. Solo puede tener una exportación predeterminada en un archivo.

### Ejemplo

message.js

```
const message = () => {  
  const name = "Jesse";  
  const age = 40;  
  return name + ' is ' + age + 'years old.';  
};  
  
export default message;
```

## Importar

Puede importar módulos a un archivo de dos maneras, en función de si se denominan exportaciones o exportaciones predeterminadas.

Las exportaciones con nombre deben desestructurarse utilizando llaves. Las exportaciones predeterminadas no.

### Ejemplo

Importe exportaciones con nombre desde el archivo person.js:

```
import { name, age } from "./person.js";
```

### Ejemplo

Importe una exportación predeterminada desde el archivo message.js:

```
import message from "./message.js";
```





## Tema 1: Introducción a ReactJS

### 3.9. Operador ternario de React ES6

El operador ternario es un operador condicional simplificado como if/

else. Sintaxis: `condition ? <expression if true> : <expression if false>`

Aquí hay un ejemplo usando if/ else:

#### Ejemplo

Antes:

```
if (authenticated) {  
  renderApp();  
} else {  
  renderLogin();  
}
```

Aquí está el mismo ejemplo usando un operador ternario:

#### Ejemplo

Con Ternario

```
authenticated ? renderApp() : renderLogin();
```

## 4. React render HTML

El objetivo de React es, en muchos sentidos, representar HTML en una página web.

React representa HTML en la página web mediante el uso de una función llamada **createRoot()** y su método **render()**.

### 4.1. Function createRoot

La función **createRoot()** toma un argumento, un elemento HTML.

El propósito de la función es definir el elemento HTML donde se debe mostrar un componente React.

### 4.2. El método de render

Luego se llama al método **render()** para definir el componente React que se debe representar.

¿Pero renderizar dónde?

Hay otra carpeta en el directorio raíz de su proyecto React, llamada "public". En esta carpeta, hay un archivo `index.html`.

Notarás un solo `<div>` en el cuerpo de este archivo. Aquí es donde se representará nuestra aplicación React.

#### Ejemplo

Mostrar un párrafo dentro de un elemento con el id de "root":

```
const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(<p>Hello</p>);
```

El resultado se muestra en el elemento `<div id="root">`:

```
<body>
  <div id="root"></div>
</body>
```

Tenga en cuenta que la identificación del elemento no tiene que llamarse "root", pero esta es la convención estándar.

## 4.3. El código HTML

El código HTML de este tutorial utiliza JSX, que le permite escribir etiquetas HTML dentro del código

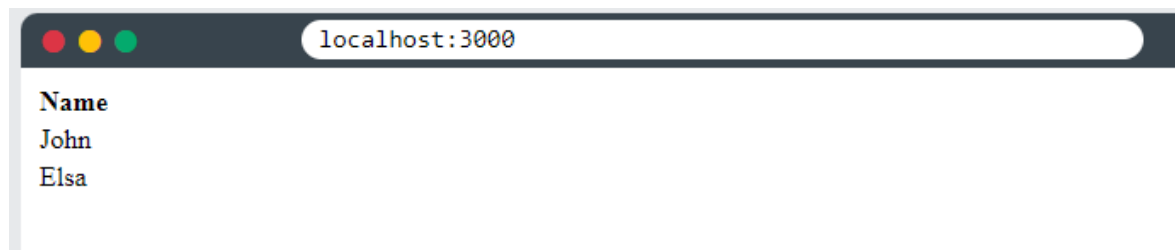
JavaScript:Ejemplo

Cree una variable que contenga código HTML y muéstrela en el nodo "root":

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

const myelement = (
  <table>
    <tr>
      <th>Name</th>
    </tr>
    <tr>
      <td>John</td>
    </tr>
    <tr>
      <td>Elsa</td>
    </tr>
  </table>
);

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(myelement);
```



# Tema 1: Introducción a ReactJS

## 4.4. El nodo root

El nodo root es el elemento HTML en el que desea mostrar el resultado. Es como un contenedor de contenido administrado por React.

NO tiene que ser un elemento **<div>** y NO tiene que tener

**id='root': Ejemplo**

El nodo raíz se puede llamar como quieras:

```
<body>

  <header id="sandy"></header>

</body>
```

Mostrar el resultado en el elemento **<header id="sandy">**:

```
const container = document.getElementById('sandy');
const root = ReactDOM.createRoot(container);
root.render(<p>Hallo</p>);
```

## 5. React JSX

### 5.1. ¿Qué es JSX?

- JSX significa JavaScript XML.
- JSX nos permite escribir HTML en React.
- JSX facilita escribir y agregar HTML en React.

### 5.2. Codificación JSX

- JSX nos permite escribir elementos HTML en JavaScript y colocarlos en el DOM sin ningún método `createElement()` y/o `appendChild()`.
- JSX convierte etiquetas HTML en elementos de react.
- No es necesario que use JSX, pero JSX facilita la escritura de aplicaciones React.

Aquí hay dos ejemplos. El primero usa JSX y el segundo no:

#### Ejemplo 1

JSX:

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

const myElement = <h1>I Love JSX!</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

#### Ejemplo 2

Sin JSX:

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

const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Como puede ver en el primer ejemplo, JSX nos permite escribir HTML directamente dentro del código JavaScript.

JSX es una extensión del lenguaje JavaScript basado en ES6 y se traduce a JavaScript normal en tiempo de ejecución.

## 5.3. Expresiones en JSX

Con JSX puedes escribir expresiones dentro de llaves { }.

La expresión puede ser una variable o propiedad de React o cualquier otra expresión de JavaScript válida. JSX ejecutará la expresión y devolverá el resultado:

### Ejemplo

Ejecutar la expresión  $5 + 5$ :

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

const myElement = <h1>React is {5 + 5} times better with JSX</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```



## 5.4. Insertar un bloque grande de HTML

Para escribir HTML en varias líneas, coloque el HTML entre paréntesis:

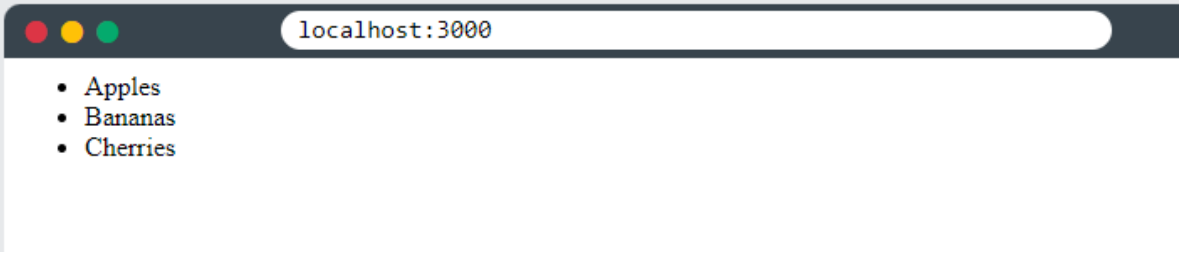
### Ejemplo

Cree una lista con tres elementos de lista:

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

const myElement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

- 
- A screenshot of a web browser window. The address bar shows 'localhost:3000'. The page content displays a bulleted list with three items: 'Apples', 'Bananas', and 'Cherries'.
- Apples
  - Bananas
  - Cherries

## Tema 1: Introducción a ReactJS

### 5.5. Un elemento de nivel superior

El código HTML debe estar envuelto en UN elemento de nivel superior.

Entonces, si desea escribir dos párrafos, debe colocarlos dentro de un elemento principal, como un elemento `div`.

Eje

mpl

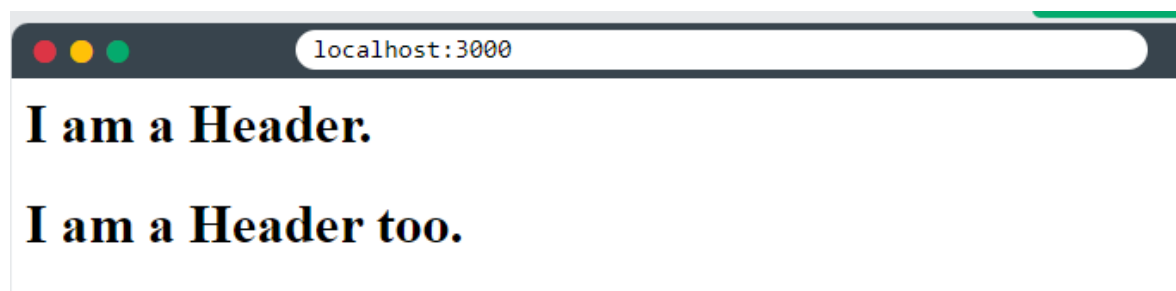
o

Envuelva dos párrafos dentro de un elemento DIV:

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

const myElement = (
  <div>
    <h1>I am a Header.</h1>
    <h1>I am a Header too.</h1>
  </div>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```



JSX arrojará un error si el HTML no es correcto o si el HTML pierde un elemento principal.

Alternativamente, puede usar un "fragmento" para envolver varias líneas. Esto evitará agregar innecesariamente nodos adicionales al DOM.

Un fragmento parece una etiqueta HTML vacía: `<></>`.



# Tema 1: Introducción a ReactJS



## Ejemplo

Envuelve dos párrafos dentro de un fragmento:

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

const myElement = (
  <>
    <p>I am a paragraph.</p>
    <p>I am a paragraph too.</p>
  </>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```



## Los elementos deben estar cerrados

JSX sigue las reglas XML y, por lo tanto, los elementos HTML deben cerrarse

correctamente. Ejemplo

Cierra los elementos vacíos con</>

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

const myElement = <input type="text" />;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

JSX arrojará un error si el HTML no se cierra correctamente.



## 5.6. Attribute class = className

El atributo **class** es un atributo muy utilizado en HTML, pero dado que JSX se representa como JavaScript y la palabra clave **class** es una palabra reservada en JavaScript, no puede utilizarla en JSX.

Utilice el atributo **className** en su lugar.

JSX resolvió esto usando **className** en su lugar. Cuando se procesa JSX, traduce **className** los atributos en atributos **class**.

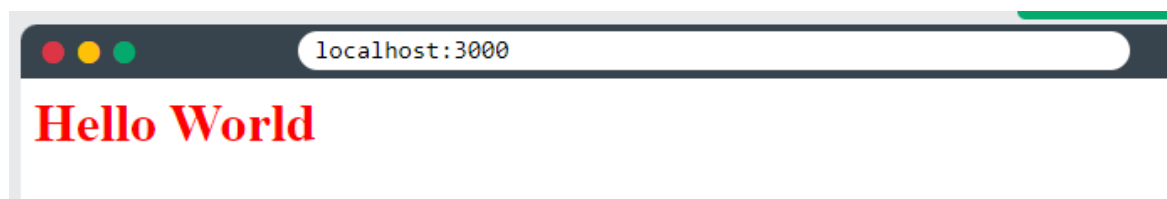
### Ejemplo

Utilice el atributo **className** en lugar de **class** en JSX:

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

const myElement = <h1 className="myclass">Hello World</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```



## Tema 1: Introducción a ReactJS

### 5.7. Condiciones - sentencias if

React admite declaraciones **if**, pero no dentro de JSX.

Para poder usar declaraciones condicionales en JSX, debe colocar las declaraciones **if** fuera de JSX, o podría usar una expresión ternaria en su lugar:

**Opción 1:** Escriba declaraciones **if** fuera del código JSX:

#### Ejemplo

Escribe "Hola" si x es menor a 10, de lo contrario "Adiós":

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

const x = 5;
let text = "Goodbye";
if (x < 10) {
  text = "Hello";
}

const myElement = <h1>{text}</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

**Opción 2:** Utilice expresiones ternarias en su lugar:

#### Ejemplo

Escribe "Hola" si x es menor a 10, de lo contrario "Adiós":

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

const x = 5;

const myElement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Tenga en cuenta que para incrustar una expresión de JavaScript dentro de JSX, el JavaScript debe estar envuelto con llaves, {}.

## 6. Componentes en ReactJS (1 hora)

### 6.1. Concepto de componentes y su importancia en ReactJS.

En ReactJS, los componentes son bloques de construcción fundamentales para crear interfaces de usuario reutilizables y modulares. Representan partes independientes y autónomas de la interfaz de usuario, que encapsulan tanto la estructura HTML como la lógica relacionada.

Un componente en ReactJS puede ser una pequeña pieza de la interfaz de usuario, como un botón o un campo de entrada, o puede ser una estructura más compleja, como una barra lateral, una lista de elementos o incluso una página completa. Los componentes pueden contener otros componentes, lo que permite una jerarquía y composición flexible de elementos de interfaz de usuario.

Aquí hay algunas características importantes de los componentes en ReactJS:

1. **Reutilización:** Los componentes en ReactJS están diseñados para ser reutilizables. Puedes crear un componente una vez y utilizarlo en diferentes partes de tu aplicación, evitando así la duplicación de código y facilitando el mantenimiento.
2. **Modularidad:** Los componentes en ReactJS promueven la creación de una arquitectura modular. Cada componente se enfoca en una funcionalidad específica, lo que mejora la legibilidad y la estructura general del código.
3. **Independencia:** Los componentes en ReactJS son autónomos y pueden funcionar de manera independiente. Tienen su propio estado interno y pueden recibir datos externos a través de props. Esto facilita la gestión y la reutilización de los componentes en diferentes contextos.
4. **Composición:** Los componentes en ReactJS se pueden combinar para crear estructuras más complejas. Puedes anidar componentes dentro de otros componentes, lo que permite una jerarquía clara y una composición flexible de la interfaz de usuario.
5. **Mantenimiento y actualización:** Debido a su naturaleza modular, los componentes en ReactJS facilitan el mantenimiento y la actualización de la aplicación. Puedes realizar cambios en un componente específico sin afectar otros componentes, lo que agiliza el proceso de desarrollo y depuración.

La importancia de los componentes en ReactJS radica en su capacidad para crear interfaces de usuario flexibles, escalables y fáciles de mantener. Los componentes promueven la reutilización de código, la modularidad y la composición, lo que mejora la eficiencia del desarrollo y la legibilidad del código. Al dividir la interfaz de usuario en componentes independientes, ReactJS permite un enfoque más modular y organizado para construir aplicaciones web interactivas.

## 6.2. Componentes funcionales vs. componentes de clase.

En ReactJS, existen dos tipos principales de componentes: componentes funcionales y componentes de clase. Cada tipo tiene sus propias características y enfoques para la creación de componentes.

### Componentes Funcionales:

- Los componentes funcionales son simplemente funciones de JavaScript.
- Se definen como una función de JavaScript que recibe props como argumento y devuelve elementos JSX que representan la interfaz de usuario.
- No tienen estado interno ni métodos de ciclo de vida propios.
- Son más sencillos y concisos de escribir, lo que los hace más legibles y fáciles de entender.
- Desde la introducción de React Hooks en ReactJS 16.8, los componentes funcionales pueden tener su propio estado y utilizar características de ciclo de vida mediante el uso de hooks como useState, useEffect, useContext, entre otros.
- Los componentes funcionales se consideran una mejor práctica en ReactJS debido a su simplicidad y la facilidad para trabajar con ellos.

### Ejemplo de componente funcional en ReactJS:

```
import React from 'react';
const MiComponenteFuncional = (props) => {
  return (
    <div>
      <h1>{props.titulo}</h1>
      <p>{props.contenido}</p>
    </div>
  );
};

export default MiComponenteFuncional;
```

## Componentes de Clase:

- Los componentes de clase son clases de JavaScript que extienden la clase base `React.Component` proporcionada por React.
- Se definen mediante la creación de una clase que contiene métodos especiales, como `render()` para definir el contenido del componente y otros métodos de ciclo de vida, como `componentDidMount()`, `componentDidUpdate()`, etc.
- Pueden tener su propio estado interno, que se maneja mediante `this.state` y se actualiza mediante `this.setState()`.
- Proporcionan más control sobre el ciclo de vida del componente y permiten realizar acciones personalizadas en diferentes etapas del ciclo de vida.
- Antes de los Hooks, los componentes de clase eran la forma principal de trabajar con el estado y los métodos de ciclo de vida en ReactJS.

## Ejemplo de componente de clase en ReactJS:

```
import React from 'react';

class MiComponenteDeClase extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.titulo}</h1>
        <p>{this.props.contenido}</p>
      </div>
    );
  }
}

export default MiComponenteDeClase;
```

En general, se recomienda utilizar componentes funcionales siempre que sea posible, ya que son más sencillos y fáciles de trabajar. Sin embargo, los componentes de clase siguen siendo útiles en ciertos casos, especialmente cuando se requiere un control más preciso sobre el ciclo de vida del componente y la gestión del estado.

## 6.3. Creación de componentes funcionales y de clase.

Aquí tienes ejemplos de cómo crear tanto componentes funcionales como componentes de clase en ReactJS:

### Componente Funcional:

```
import React from 'react';

const MiComponenteFuncional = (props) => {
  return (
    <div>
      <h1>{props.titulo}</h1>
      <p>{props.contenido}</p>
    </div>
  );
};

export default MiComponenteFuncional;
```

En este ejemplo, creamos un componente funcional llamado `MiComponenteFuncional` que recibe props como argumento y devuelve elementos JSX. En este caso, renderizamos un título y un contenido pasado a través de las props.

### Componente de Clase:

```
import React from 'react';

class MiComponenteDeClase extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.titulo}</h1>
        <p>{this.props.contenido}</p>
      </div>
    );
  }
}

export default MiComponenteDeClase;
```

# Tema 1: Introducción a ReactJS



En este ejemplo, creamos un componente de clase llamado `MiComponenteDeClase` que extiende la clase `baseReact.Component`. Definimos un método `render()` en la clase, que devuelve elementos JSX. Al igual que en el ejemplo anterior, renderizamos un título y un contenido pasado a través de las props.

En ambos casos, estos componentes se pueden utilizar en otros componentes o en el punto de entrada de la aplicación de React para renderizarlos en la interfaz de usuario. Por ejemplo:

```
import React from 'react';
import ReactDOM from 'react-dom';
import MiComponenteFuncional from './MiComponenteFuncional';
import MiComponenteDeClase from './MiComponenteDeClase';

ReactDOM.render(
  <div>
    <MiComponenteFuncional titulo="Componente Funcional" contenido="Este es un componente funcional." />
    <MiComponenteDeClase titulo="Componente de Clase" contenido="Este es un componente de clase." />
  </div>,
  document.getElementById('root')
);
```

En este caso, estamos renderizando ambos componentes (`MiComponenteFuncional` y `MiComponenteDeClase`) en el punto de entrada de la aplicación de React (`document.getElementById('root')`). Les pasamos props a ambos componentes para personalizar su contenido.

Recuerda que, en ReactJS, puedes utilizar tanto componentes funcionales como componentes de clase según tus necesidades y preferencias. Los componentes funcionales son más simples y fáciles de trabajar, mientras que los componentes de clase proporcionan más control sobre el ciclo de vida del componente y la gestión del estado.





## 6.4. Renderización de elementos JSX en los componentes.

En ReactJS, la renderización de elementos JSX es un paso fundamental en la creación de componentes. JSX es una extensión de sintaxis de JavaScript que permite escribir código HTML similar dentro de los componentes de React.

Aquí tienes un ejemplo de cómo renderizar elementos JSX dentro de un componente:

```
import React from 'react';

const MiComponente = () => {
  return (
    <div>
      <h1>Título del componente</h1>
      <p>Contenido del componente</p>
    </div>
  );
};

export default MiComponente;
```

En este ejemplo, el componente `MiComponente` es un componente funcional que devuelve un elemento JSX. Dentro del bloque de retorno, hemos creado una estructura de elementos HTML, incluyendo un encabezado (`<h1>`) y un párrafo (`<p>`). Estos elementos JSX se renderizarán en la interfaz de usuario cuando el componente sea utilizado.

Puedes incluir cualquier elemento HTML válido dentro del componente y utilizar expresiones de JavaScript dentro de las llaves (`{}`) para insertar valores dinámicos o evaluar lógica. Por ejemplo:

```
import React from 'react';
const MiComponente = () => {
  const nombre = 'Juan';
  const edad = 25;

  return (
    <div>
      <h1>Título del componente</h1>
      <p>Nombre: {nombre}</p>
      <p>Edad: {edad}</p>
    </div>
  );
};

export default MiComponente;
```

# Tema 1: Introducción a ReactJS



En este caso, hemos definido dos variables (nombre y edad) y las hemos utilizado dentro de los elementos JSX para mostrar valores dinámicos. Las llaves ({} ) permiten insertar expresiones de JavaScript dentro del código JSX.

Recuerda que cada componente de React puede tener solo un elemento raíz en el bloque de retorno. Por lo tanto, si deseas renderizar varios elementos, asegúrate de envolverlos en un contenedor (como un `<div>`) o utilizar fragmentos (`<React.Fragment>`) para evitar un elemento adicional en el DOM.

En resumen, la renderización de elementos JSX dentro de los componentes de React es una forma intuitiva y poderosa de construir interfaces de usuario. Puedes crear estructuras HTML complejas, utilizar expresiones de JavaScript y mostrar valores dinámicos dentro de los elementos JSX para personalizar y adaptar la interfaz de usuario según tus necesidades.



## 6.5. RESUMEN

### Ejemplo

Crear un componente Function llamado Car

```
function Car() {  
  return <h2>Hi, I am a Car!</h2>;  
}
```

### Renderizando de un componente

Ahora su aplicación React tiene un componente llamado Car, que devuelve un elemento.<h2>

Para utilizar este componente en la aplicación, utilice una sintaxis similar a la HTML normal:

<Car />Ejemplo

Muestre el componente en el elemento "raíz":Car

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
  
function Car() {  
  return <h2>Hi, I am a Car!</h2>;  
}  
  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(<Car />);
```

# Tema 1: Introducción a ReactJS



## Props

Los componentes se pueden pasar como , que significa propiedades.props

Los accesorios son como argumentos de función, y se envían al componente como

atributos.Ejemplo

Utilice un atributo para pasar un color al componente Coche y utilícelo en el cuadro Función render():

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

function Car(props) {
  return <h2>I am a {props.color} Car!</h2>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car color="red"/>);
```

## Componentes en componentes

Podemos referirnos a componentes dentro de otros

componentes:Ejemplo

Utilice el componente Coche dentro del componente Garaje:

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

function Car() {
  return <h2>I am a Car!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my Garage?</h1>
      <Car />
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);
```



# Tema 1: Introducción a ReactJS



## Componentes en archivos

React se trata de reutilizar el código, y se recomienda dividir sus componentes en archivos separados. Para hacer eso, cree un nuevo archivo con una extensión de archivo y coloque el código dentro de él: .js. Tenga en cuenta que el nombre de archivo debe comenzar con un carácter en mayúsculas.

### Ejemplo

Este es el nuevo archivo, lo llamamos "Car.js":

```
function Car() {  
  return <h2>Hi, I am a Car!</h2>;  
}  
  
export default Car;
```

Para poder utilizar el componente Coche, debe importar el archivo en su aplicación.

### Ejemplo

Ahora importamos el archivo "Car.js" en la aplicación, y podemos usar el componente como si se hubiera creado aquí. Car

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import Car from './Car.js';  
  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(<Car />);
```



## 7. Ejercicio\_1: Creación de un componente simple

### Creación de un componente básico utilizando lo aprendido en la sesión.

A continuación, te guiaré en la creación de un componente básico utilizando lo aprendido en la sesión:

Crea un nuevo archivo para tu componente. Puedes nombrarlo como desees y asegúrate de que tenga la extensión .js o .jsx. Por ejemplo, MiComponente.js.

En el archivo de tu componente, importa la biblioteca de React y cualquier otra dependencia necesaria:

```
import React from 'react';

const MiComponente = () => {
  return (
    <div>
      <h1>Título del componente</h1>
      <p>Contenido del componente</p>
    </div>
  );
};
```

En este ejemplo, el componente MiComponente devuelve una estructura básica de elementos JSX que incluye un encabezado (<h1>) y un párrafo (<p>). Siéntete libre de personalizar el contenido y la estructura según tus necesidades.

Exporta el componente para poder utilizarlo en otros lugares de tu aplicación:

```
export default MiComponente;
```

Con esto, el componente MiComponente está listo para ser utilizado en otros componentes o en el punto de entrada de tu aplicación.

Recuerda que, para utilizar este componente, debes importarlo en el archivo correspondiente y luego incluirlo en la estructura de elementos JSX donde desees que se renderice. Por ejemplo:

```
import React from 'react';
import ReactDOM from 'react-dom';
import MiComponente from './MiComponente';

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

# Tema 1: Introducción a ReactJS



En este caso, estamos utilizando `ReactDOM.render()` para renderizar el componente `MiComponente` en elemento con el ID `'root'`.

Puedes personalizar tu componente y su estructura JSX según tus necesidades. Utiliza los conocimientos adquiridos en la sesión para agregar elementos adicionales, modificar el contenido, utilizar props, realizar cálculos y cualquier otra personalización que desees.

