Docs_

Tutorial ReactJS

Sesión 1: Introducción a ReactJS

Duración: 5 horas

- Introducción a ReactJS (30 minutos)
- Configuración del entorno de desarrollo (1 hora)
- Componentes en ReactJS (1 hora)
- JSX (1 hora)
- Props (1 hora)
- Ejercicio práctico: Creación de un componente simple (30 minutos)

Introducción
Primeros Pasos
Actualizar React
React ES6
React Render HTML
React JSX
React components
React Props

Sesión 2: Estado y Ciclo de Vida de los Componentes

Duración: 5 horas

- Estado en ReactJS: Introducción al estado, actualización del estado.
- Ciclo de vida de los componentes de clase: componentDidMount, componentDidUpdate,componentWillUnmount.
- Gestión del estado y eventos: Manipulación del estado y manejo de eventos.
- Refs: Uso de refs para acceder a elementos del DOM.
- Ejercicio práctico: Creación de un contador utilizando el estado y el ciclo de vida de los componentes.

React Class
React Events
React Conditionals

Sesión 3: Comunicación entre Componentes

Duración: 5 horas

- Comunicación entre componentes: Paso de datos entre componentes (props, state lifting).
- Componentes controlados y no controlados: Diferencias y casos de uso.
- Eventos personalizados: Creación y manejo de eventos personalizados.
- Context API: Uso del Context API para compartir datos entre componentes.
- Ejercicio práctico: Construcción de un formulario con validación utilizando la comunicación entrecomponentes.

React
Lists
React
Forms

Sesión 4: Enrutamiento y Gestión de Estado Avanzada

Duración: 5 horas

- Enrutamiento en React: Introducción a React Router para la navegación entre páginas.
- Gestión de estado avanzada: Introducción a Redux para la gestión del estado global.
- Acciones, reducers y store en Redux: Conceptos básicos de Redux.
- Integración de Redux en una aplicación de React: Configuración y uso de Redux.
- Ejercicio práctico: Implementación de la navegación entre diferentes secciones de una aplicaciónutilizando React Router y Redux.

React Router
React Memo
React CSS
Styling React
Sass Styling

Sesión 5: Pruebas y Optimización

Duración: 5 horas

- Pruebas en ReactJS: Introducción a las pruebas unitarias y de integración con Jest y Enzyme.
- Buenas prácticas de rendimiento en ReactJS: Lazy loading, memoización, optimización de renderizado.
- Deploy de una aplicación de ReactJS: Opciones y consideraciones para implementar una aplicación deReact en producción.
- Ejercicio práctico: Realización de pruebas unitarias y optimización de rendimiento en una aplicación deReactJS

React Hooks

What is a
Hook?
useState
useEffect
useContext
useRef
useReducer
useCallback
useMemo
Custom
Hooks

Introducción

¿Qué es React?

React, a veces denominado framework de JavaScript frontend, es una biblioteca de JavaScript creada porFacebook.

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, donderealiza 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.

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 en

HTML.Pero para usar React en producción, necesita npm y Node.js instalados.

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, yel 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>;
```

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

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 qye npm?

No, npx no es lo mismo que npm, aunque ambos son herramientas relacionadas con Node.js y el ecosistema deJavaScript. 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 scriptspersonalizados 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 instalapaquetes 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 deinstalación previa.

Si anteriormente realizó una instalación global **create-react-app**, se recomienda que desinstale el paquete paraasegurarse 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
```


Ejecute la aplicación React

¡Ahora está listo para ejecutar su primera aplicación React

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

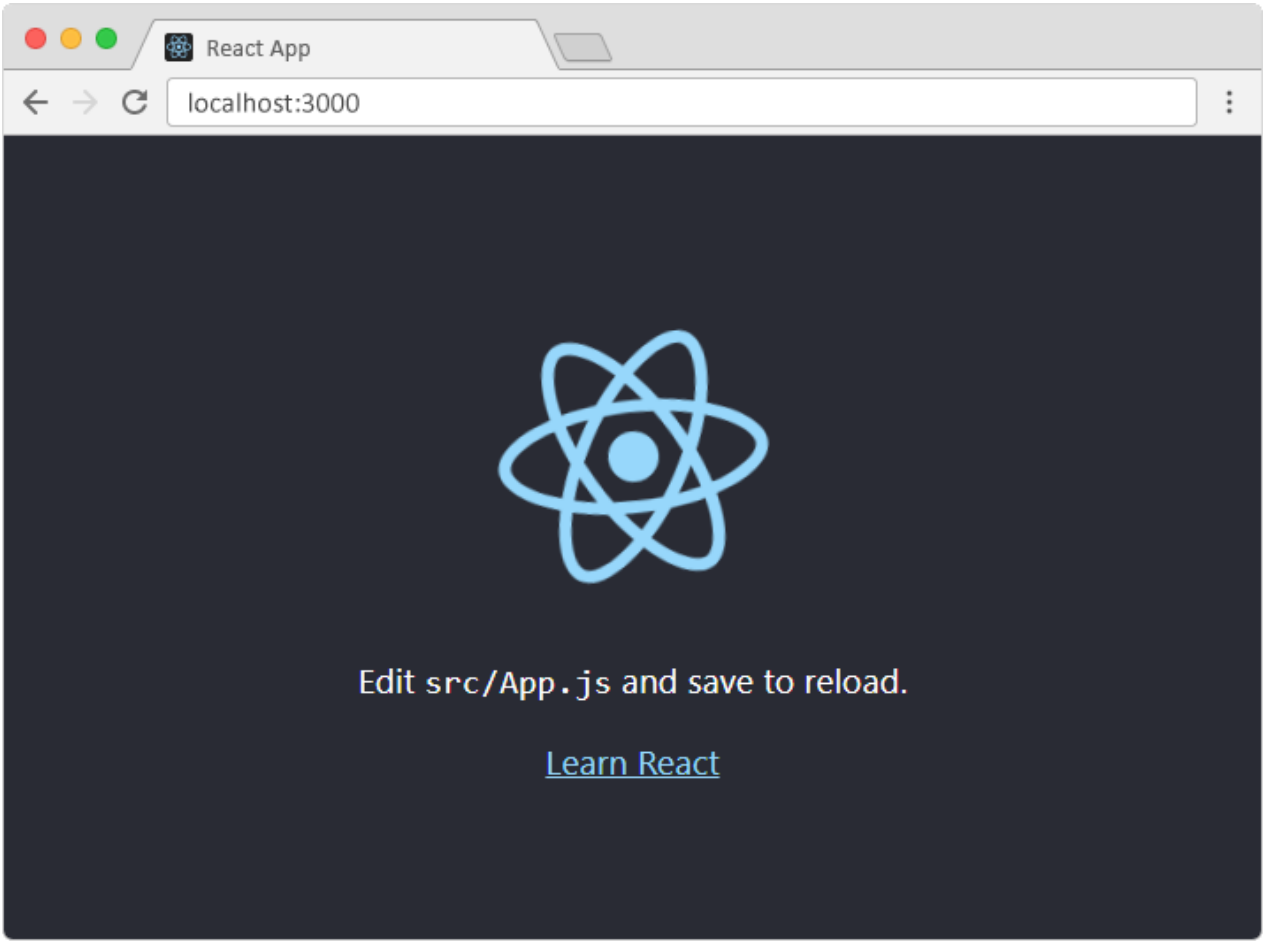
react-app:

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

```
npm start
```

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

El resultado:



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 archivollamado 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"

    `
```

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!

Ejemplo 2

Reemplace todo el contenido dentro de ** ** con un elemento

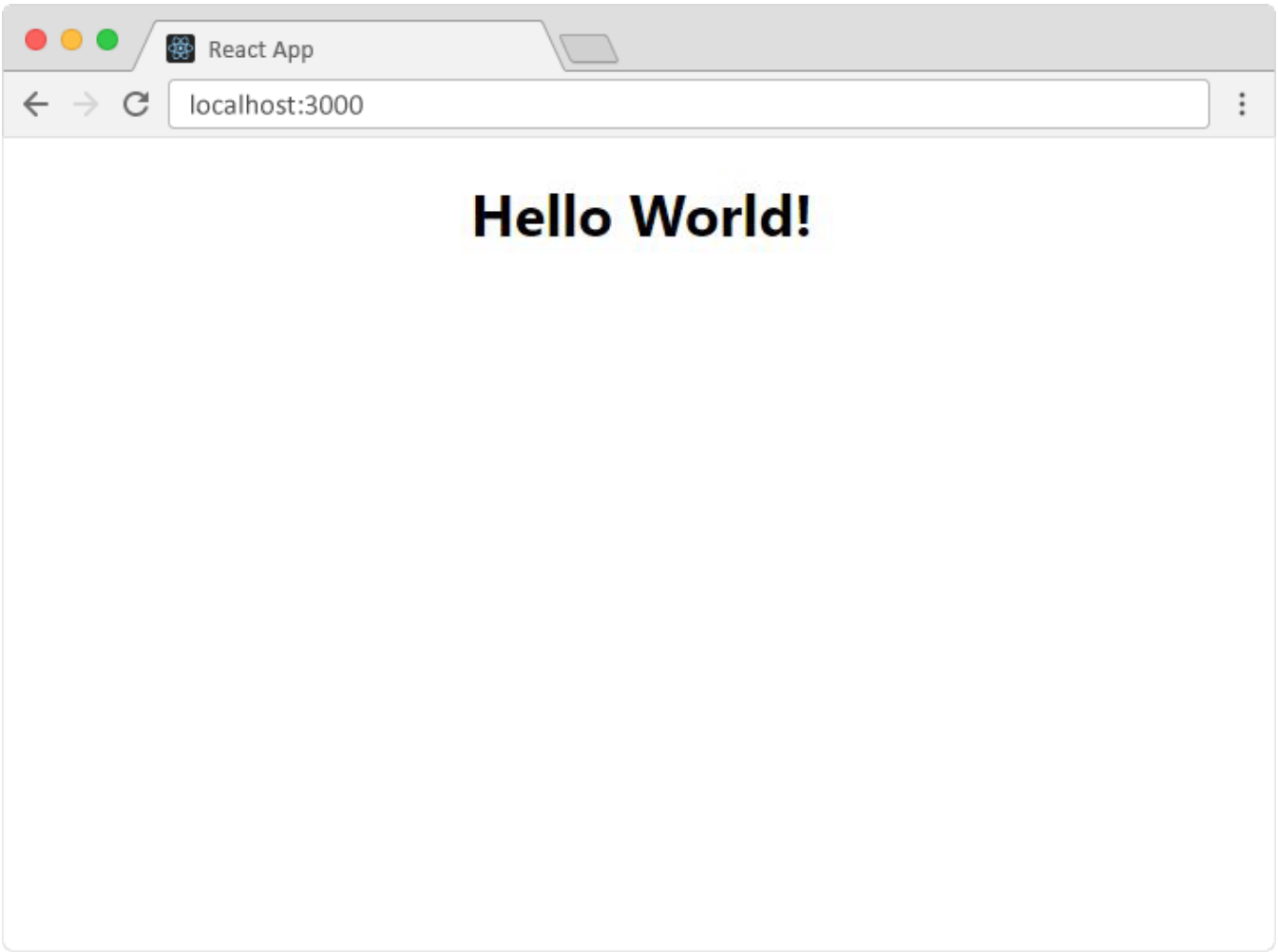
<h1>.Vea los cambios en el navegador cuando haga clic en Guardar.

```
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:



Seguimos

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

Procederemos a desmontar la carperta 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);


/*
```


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 delcliente.

```
// 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 secomportará como React 17.

React ES6

¿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 ytambién 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 mimsmo 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 oES2015). 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 ylegible.

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 estadoglobal.

En resumen, React ES6 se refiere al uso de React con las características y sintaxis de ECMAScript 6, mientras que ReactJS esla 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 ES6puede hacer que el código sea más moderno y expresivo.

- ## 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 palabraclave **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. Estaes 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.

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).

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 obtenemosacceso a las propiedades y métodos del padre.

- ## 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;
```

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 laventana, 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 elusuario 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 segundoejemplo devuelve el objeto de encabezado dos veces.

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);
```

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 regulareses lo que desea, si no, use funciones de flecha.

- **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 noestuvieran 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 .

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

constantePero puedes:

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

- ## 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 matrizcomo 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);
```

- **React ES6 Desestructuración**

desestructuración

Para ver la desestructuración, haremos un sándwich. ¿Saca todo de la nevera para hacer su bocadillo? No, solosaca 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 queestemos 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;
```

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 + '.';
}
```

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 luegousando 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 + '.';
```

- ## 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];
```

1,2,3,4,5,6

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>");
```

1

2

3,4,5,6

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 fuesobrescrita por el último objeto que se pasó, updateMyVehicle. El color resultante ahora es amarillo.

- **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 parteinferior.

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 }
```


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 oexportaciones 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";
```


- ## 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();
```


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().**

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.

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 archivoindex.html.

Notarás un solo 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 :

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

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

El código HTML

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

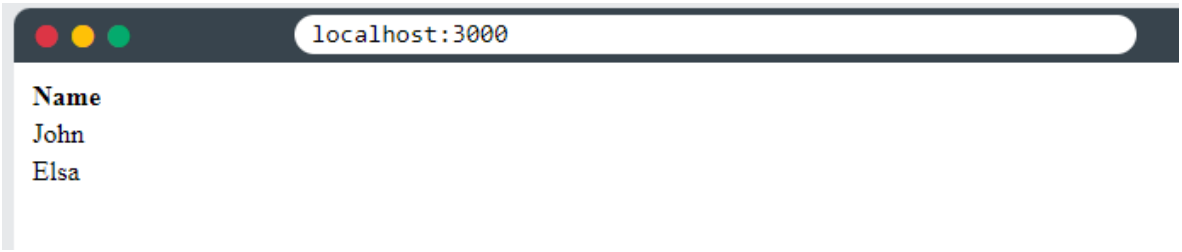
JavaScript:Ejemplo

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);
```



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 ** ** 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>);
```


React JSX

¿Qué es JSX?

JSX significa JavaScript XML.

JSX nos permite escribir HTML en React.

JSX facilita escribir y agregar HTML en React.

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.

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. JSXejecutará 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);
```


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);
```

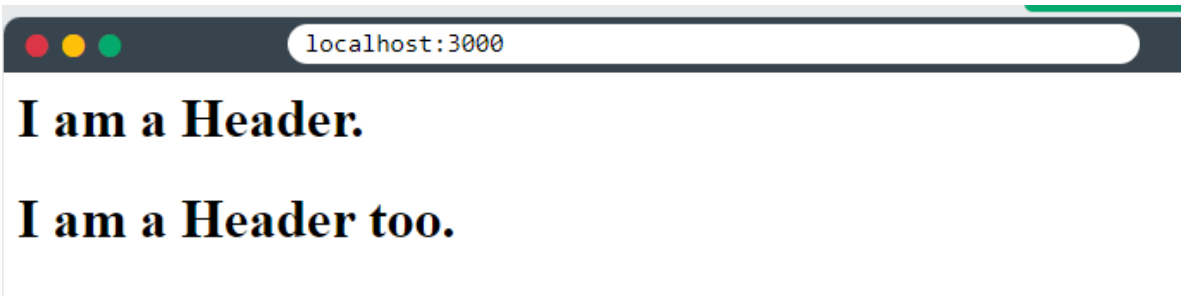
Un elemento de nivel superior

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.*

Ejemp

Envuelva dos párrafos dentro de un elemento DIV:



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 innecesariamentenodos adicionales al DOM.

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

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);
```

```
import React from 'react';
import ReactDOM from 'react-do/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);
```

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.

Attribute class = className

El atributo **class** es un atributo muy utilizado en HTML, pero dado que JSX se representa como JavaScript y lapalabra 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 enatributos **class**.

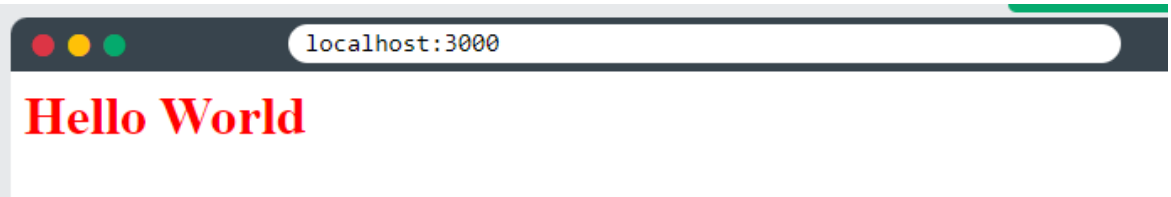
Ejemplo

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);
```



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 usaruna 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);
```


Opcion 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 envueltocon llaves, {}.

Componentes de React

Los componentes son como funciones que devuelven elementos HTML.

Componentes de React

Los componentes son bits de código independientes y reutilizables. Tienen el mismo propósito que las funcionesde JavaScript, pero trabajar de forma aislada y devolver HTML.

Los componentes vienen en dos tipos, componentes de clase y componentes de función, en este tutorial nos concentraremos en los componentes de la función.

En las bases de código de React más antiguas, puede encontrar componentes de clase utilizados principalmente.Ahora se sugiere usar componentes de función junto con ganchos, que se agregaron en React 16.8. Hay una sección opcional sobre componentes de clase para su referencia.

Cree su primer componente

Al crear un componente React, el nombre del componente DEBE comenzar con un letra mayúscula.

Componente de clase

Un componente de clase debe incluir la instrucción. Esta instrucción crea una herencia a React.Component yproporciona al componente acceso a las funciones de React.Component.extends React.Component

El componente también requiere un método, este método devuelve HTML.render()

Ejemplo

Crear un componente Class denominado Car

```
class Car extends React.Component {
 render() {
  return <h2>Hi, I am a Car!</h2>;
 }
}
```


Componente de función

Aquí está el mismo ejemplo que el anterior, pero creado usando un componente Function en su lugar.

Un componente Function también devuelve HTML y se comporta de la misma manera que un componente Class, pero los componentes de la función se pueden escribir usando mucho menos código, son más fáciles deentender y serán preferidos en este tutorial.

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>;
}
```

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 />);
```

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:.jsTenga 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 creadoaquí.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 />);
```

React Props

Los accesorios son argumentos pasados a los componentes de

React. Los accesorios se pasan a los componentes a través de

atributos HTML.props Representa propiedades.

React Props

Los React Props son como argumentos de función en JavaScript y atributos en

HTML. Para enviar props a un componente, utilice la misma sintaxis que los

atributos HTML:Ejemplo

Agregue un atributo "brand" al elemento Car:

```
const myElement = <Car brand="Ford" />;
```

El componente recibe el argumento como un

objeto:propsEjemplo

Utilice el atributo de marca en el componente:

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

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

const myElement = <Car brand="Ford" />;
```

Pasar datos

Los accesorios también son la forma en que pasas datos de un componente a otro, como

parámetros.Ejemplo

Envíe la propiedad "marca" del componente Garaje al componente Coche:

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

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

function Garage() {
  return (
    <>
            <h1>Who lives in my garage?</h1>
            <Car brand="Ford" />
    </>
  );
}

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

Si tiene una variable para enviar, y no una cadena como en el ejemplo anterior, simplemente coloque el nombrede la variable entre llaves:

Ejemplo

Cree una variable denominada y envíela al componente:carNameCar

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

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

function  Garage()  {
  const carName = "Ford";
  return (
    <>
            <h1>Who lives in my garage?</h1>
            <Car brand={ carName } />
    </>
  );
}

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

O si era un objeto:

Ejemplo

Cree un objeto denominado y envíelo al componente:carInfoCar

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

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

function Garage() {
  const carInfo = { name: "Ford", model: "Mustang" };
  return (
    <>
            <h1>Who lives in my garage?</h1>
            <Car brand={ carInfo } />
    </>
  );
}

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

Nota: ¡Los accesorios de React son de solo lectura! Obtendrá un error si intenta cambiar su valor.

React Events

Al igual que los eventos HTML DOM, React puede realizar acciones basadas en eventos de

usuario.React tiene los mismos eventos que HTML: clic, cambio, mouseover, etc.

Agregar eventos

Los eventos React están escritos en sintaxis camelCase:

onClick En lugar de **onclick**

Los controladores de eventos de React se escriben dentro de{ }:

onClick={shoot} En lugar de **onClick="shoot()"**

ReactJS

```
<button onClick={shoot}>Take the Shot!</button>
```

.HTML:

```
<button onclick="shoot()">Take the Shot!</button>
```

Ejemplo:

Coloque la función shoot dentro del componente Football:

```
import React from 'react';

import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }

  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}

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

Argumentos de paso

Para pasar un argumento a un controlador de eventos, utilice una función de flecha.

Ejemplo:

Enviar "¡Objetivo!" como parámetro a la función, usando la flecha función:shoot

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

function Football() {
  const shoot = (a) => {
    alert(a);
  }

  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}

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

React Event (objeto)

Los controladores de eventos tienen acceso al evento React que activó la

función.En nuestro ejemplo, el evento es el evento "clic".

Ejemplo:

Función de flecha: Envío manual del objeto de evento:

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

function Football() {
  const shoot = (a, b) => {
    alert(b.type);
                /*
                'b' represents the React event that triggered the function.
    In this case, the 'click' event
                */
  }

  return (
    <button onClick={(event) => shoot("Goal!", event)}>Take the shot!</button>
  );
}

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

React Router

Create React App no incluye enrutamiento de

página.React Router es la solución más popular.

Agregar React Router

Para agregar React Router en su aplicación, ejecute esto en el terminal desde el directorio raíz de la aplicación:

```
npm i -D react-router-dom
```

Si está actualizando desde v5, deberá usar el indicador @latest:

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

Estructura de carpetas

Para crear una aplicación con varias rutas de página, comencemos primero con la estructura de

archivos.Dentro de la carpeta src, crearemos una carpeta nombrada pages con varios archivos:

```
src\pages\:

Layout.js
Home.js
Blogs.js
Contact.js
NoPage.js
```

Cada archivo contendrá un componente React muy básico.

Uso básico

Ahora usaremos nuestro Router en nuestro archivo.index.js

Ejemplo

Use React Router para enrutar a páginas basadas en

URL:index.js:

```
import ReactDOM from "react-dom/client";

import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";

import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";


export default function App() {
  return (

    <BrowserRouter>

      <Routes>

        <Route path="/" element={<Layout />}>

          <Route index element={<Home />} />

          <Route path="blogs" element={<Blogs />} />

          <Route path="contact" element={<Contact />} />

          <Route path="*" element={<NoPage />} />

        </Route>

      </Routes>

    </BrowserRouter>

  );

}


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

Ejemplo explicado

Primero envolvemos nuestro contenido con .<BrowserRouter>

Luego definimos nuestro archivo .<Routes>. Una aplicación puede tener varios archivos<Routes> . Nuestroejemplo básico solo usa uno

<Route>s se puede anidar. El primero.<Route>/ tiene una ruta de y renderiza el Layout componente

Las <Route> anidadas heredan y se agregan a la ruta principal. Así que la blogs ruta se combina con el padre y seconvierte en . /blogs

La ruta Home del componente no tiene una ruta pero tiene un atributo Index. Eso especifica esta ruta como laruta predeterminada para la ruta principal, que es /

Establecer el Path a * actuará como un catch-all para cualquier URL no definida. Esto es ideal para una página deerror 404.

Pages / Components

El componente Layout tiene elementos <Outlet> y

<Link>El<Outlet> representa la ruta actual

seleccionada.

<Link> se utiliza para establecer la URL y realizar un seguimiento del historial de

navegación.Cada vez que enlazamos a una ruta interna, usaremos <link> en lugar de<a

href="">

La "ruta de diseño" es un componente compartido que inserta contenido común en todas las páginas, como unmenú de navegación.
Layout.js

```
import { Outlet, Link } from "react-router-dom";
```


```
const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />

    </>
  )

};

export default Layout;
```

Home.js:

```
const Home = () => {

  return <h1>Home</h1>;


};


export default Home;
```

Blogs.js:

```
const Blogs = () => {

  return <h1>Blog Articles</h1>;
```


```
};

export default Blogs;
```

Contact.js:

```
const Contact = () => {
  return <h1>Contact Me</h1>;
};

export default Contact;
```

NoPage.js:

```
const NoPage = () => {
  return <h1>404</h1>;
};

export default NoPage;
```