APRENDE REACT ›

Inicio rápido

¡Bienvenido a la documentación de React! Esta página te dará una introducción al 80% de los conceptos de React que usarás a diario.

Aprenderás

- Cómo crear y anidar componentes
- Cómo añadir marcado y estilos
- Cómo mostrar datos
- Cómo renderizar condicionales y listas
- Cómo responder a eventos y actualizar la pantalla
- Cómo compartir datos entre componentes

Crear y anidar componentes

Las aplicaciones de React están hechas a partir de *componentes*. Un componente es una pieza de UI (siglas en inglés de interfaz de usuario) que tiene su propia lógica y apariencia. Un componente puede ser tan pequeño como un botón, o tan grande como toda una página.

Los componentes de React son funciones de JavaScript que devuelven *markup* (marcado):

```
function MyButton() {
  return (
    <button>Soy un botón</button>
  );
}
```

Ahora que has declarado `MyButton`, puedes anidarlo en otro componente:

```
export default function MyApp() {
  return (
    <div>
      <h1>Bienvenido a mi aplicación</h1>
      <MyButton />
    </div>
  );
}
```

Nota que `<MyButton />` empieza con mayúscula. Así es como sabes que es un componente de React. Los nombres de los componentes de React siempre deben comenzar con mayúscula, mientras las etiquetas HTML deben estar minúsculas.

Mira el resultado:

App.js

Descargar Reiniciar Bifurcar

```
1  function MyButton() {
2    return (
3      <button>
4        Soy un botón
5      </button>
6    );
7  }
8
9  export default function MyApp() {
10   return (
11     <div>
12       <h1>Bienvenido a mi aplicación</h1>
```

Mostrar más

Bienvenido a mi aplicación

Soy un botón

Las palabras clave `export default` especifican el componente principal en el archivo. Si no estás familiarizado con alguna parte de la sintaxis de JavaScript, MDN y javascript.info tienen magníficas referencias.

Escribir marcado con JSX

La sintaxis de marcado que viste arriba se llama *JSX*. Es totalmente opcional, pero la mayoría de los proyectos de React usan JSX por la comodidad que ofrece. Todas las herramientas que recomendamos para el desarrollo local son compatibles con JSX sin ningún tipo de configuración.

JSX es más estricto que HTML. Tienes que cerrar etiquetas como `<br />`. Tu componente tampoco puede devolver múltiples etiquetas de JSX. Debes envolverlas en un padre compartido, como `<div>...</div>` o en un envoltorio vacío `<>...</>`:

```
function AboutPage() {
  return (
    <>
      <h1>Acerca de</h1>
      <p>Hola.<br />¿Cómo vas?</p>
    </>
  );
}
```

Si tienes mucho HTML que convertir a JSX, puedes utilizar un convertidor en línea.

Añadir estilos

En React, especificas una clase de CSS con `className`. Funciona de la misma forma que el atributo `class` de HTML:

```
<img className="avatar" />
```

Luego escribes las reglas CSS para esa clase en un archivo CSS aparte:

```
/* In your CSS */
.avatar {
  border-radius: 50%;
}
```

React no prescribe como debes añadir tus archivos CSS. En el caso más simple, añades una etiqueta `<link>` a tu HTML. Si utilizas una herramienta de construcción o un framework, consulta su documentación para saber como añadir un archivo CSS a tu proyecto.

Mostrar datos

JSX te permite poner marcado dentro de JavaScript. Las llaves te permiten «escapar de nuevo» hacia JavaScript de forma tal que puedas incrustar una variable de tu código y mostrársela al usuario. Por ejemplo,

esto mostrará `user.name`:

```
return (
  <h1>
    {user.name}
  </h1>
);
```

También puedes «escaparte hacia JavaScript» en los atributos JSX, pero tienes que utilizar llaves *en lugar de* comillas. Por ejemplo, `className="avatar"` pasa la cadena `"avatar"` como la clase CSS, pero `src={user.imageUrl}` lee el valor de la variable de JavaScript `user.imageUrl` y luego pasa el valor como el atributo `src`:

```
return (
  <img
    className="avatar"
    src={user.imageUrl}
  />
);
```

Puedes también poner expresiones más complejas dentro de llaves, por ejemplo, concatenación de cadenas:

App.js Descargar Reiniciar Bifurcar

```
const user = {
  name: 'Hedy Lamarr',
  imageUrl: 'https://i.imgur.com/yXOvdOSs.jpg',
  imageSize: 90,
};

export default function Profile() {
  return (
    <>
      <h1>{user.name}</h1>
      <img
        className="avatar"
```

Mostrar más

En el ejemplo de arriba, `style={{}}` no es una sintaxis especial, sino un objeto regular `{}` dentro de las llaves de JSX de `style={ }`. Puedes utilizar el atributo `style` cuando tus estilos dependen de variables de

JavaScript.

Renderizado condicional

En React, no hay una sintaxis especial para escribir condicionales. En cambio, usarás las mismas técnicas que usas al escribir código regular de JavaScript. Por ejemplo, puedes usar una sentencia `if` para incluir JSX condicionalmente:

```
let content;
if (isLoggedIn) {
  content = <AdminPanel />;
} else {
  content = <LoginForm />;
}
return (
  <div>
    {content}
  </div>
);
```

Si prefieres un código más compacto, puedes utilizar el operador `?` condicional. A diferencia de `if`, funciona dentro de JSX:

```
<div>
  {isLoggedIn ? (
```

```
      <AdminPanel />
    ) : (
      <LoginForm />
    )}
  </div>
```

Cuando no necesites la rama `else`, puedes también usar la sintaxis lógica `&&`, más breve:

```
<div>
  {isLoggedIn && <AdminPanel />}
</div>
```

Todos estos enfoques también funcionan para especificar atributos condicionalmente. Si no estás familiarizado con toda esta sintaxis de JavaScript, puedes comenzar por usar siempre `if...else`.

Renderizado de listas

Dependerás de funcionalidades de JavaScript como los bucles `for` y la función map() de los arreglos para renderizar listas de componentes.

Por ejemplo, digamos que tienes un arreglo de productos:

```
const products = [
  { title: 'Col', id: 1 },
  { title: 'Ajo', id: 2 },
```

```
  { title: 'Manzana', id: 3 },
];
```

Dentro de tu componente, utiliza la función `map()` para transformar el arreglo de productos en un arreglo de elementos `<li>`:

```
const listItems = products.map(product =>
  <li key={product.id}>
    {product.title}
  </li>
);

return (
  <ul>{listItems}</ul>
);
```

Nota que `<li>` tiene un atributo `key` (llave). Para cada elemento en una lista, debes pasar una cadena o un número que identifique ese elemento de forma única entre sus hermanos. Usualmente, una llave debe provenir de tus datos, como un ID de una base de datos. React dependerá de tus llaves para entender qué ha ocurrido si luego insertas, eliminas o reordenas los elementos.

App.js

Descargar Reiniciar Bifurcar

```
const products = [
  { title: 'Col', isFruit: false, id: 1 },
```

```
  { title: 'Ajo', isFruit: false, id: 2 },
  { title: 'Manzana', isFruit: true, id: 3 },
];

export default function ShoppingList() {
  const listItems = products.map(product =>
    <li
      key={product.id}
      style={{
```

Mostrar más

Responder a eventos

Puedes responder a eventos declarando funciones *controladoras de eventos* dentro de tus componentes:

```
function MyButton() {
  function handleClick() {
    alert('¡Me hiciste clic!');
  }

  return (
    <button onClick={handleClick}>
      Hazme clic
    </button>
  );
}
```

¡Nota que `onClick={handleClick}` no tiene paréntesis al final! No *llames* a la función controladora de evento: solamente necesitas *pasarla hacia abajo*. React llamará a tu controlador de evento cuando el usuario haga clic en el botón.

Actualizar la pantalla

A menudo, querrás que tu componente «recuerde» alguna información y la muestre. Por ejemplo, quizá quieras contar el número de veces que hiciste clic en un botón. Para lograrlo, añade *estado* a tu componente.

Primero, importa `useState` de React:

```
import { useState } from 'react';
```

Ahora puedes declarar una *variable de estado* dentro de tu componente:

```
function MyButton() {
  const [count, setCount] = useState(0);
  // ...
```

Obtendrás dos cosas de `useState`: el estado actual (`count`), y la función que te permite actualizarlo (`setCount`). Puedes nombrarlos de cualquier forma, pero la convención es llamarlos algo como `[something, setSomething]`.

La primera vez que se muestra el botón, `count` será `0` porque pasaste `0` a `useState()`. Cuando quieras cambiar el estado llama a `setCount()` y pásale el nuevo valor. Al hacer clic en este botón se incrementará el contador:

```
function MyButton() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      Hiciste clic {count} veces
    </button>
  );
```

```
}
```

React llamará de nuevo a la función del componente. Esta vez, `count` será `1`. Luego será `2`. Y así sucesivamente.

Si renderizas el mismo componente varias veces, cada uno obtendrá su propio estado. Intenta hacer clic independientemente en cada botón:

App.js

Descargar Reiniciar Bifurcar

```
import { useState } from 'react';

export default function MyApp() {
  return (
    <div>
      <h1>Contadores que se actualizan separadamente</h1>
      <MyButton />
      <MyButton />
    </div>
  );
}
```

Mostrar más

Nota que cada botón «recuerda» su propio estado `count` y que no afecta a otros botones.

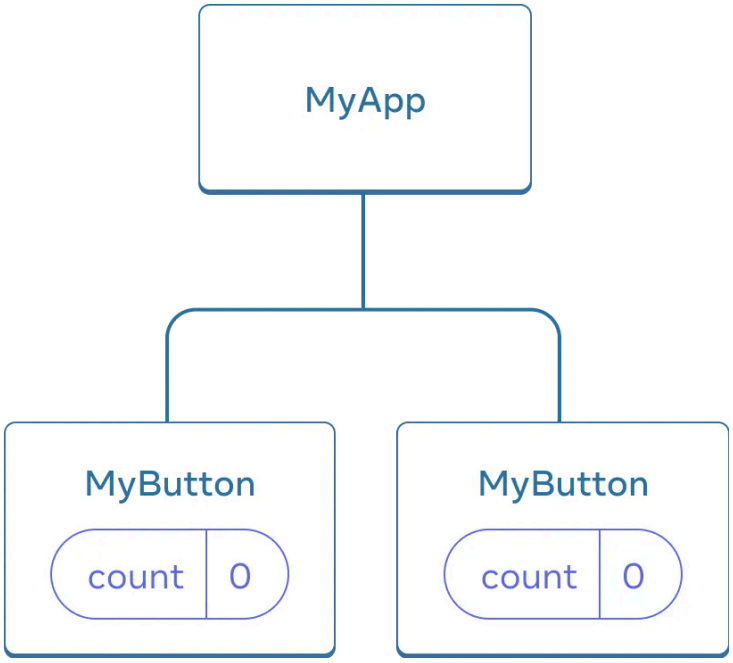
El uso de los Hooks

Las funciones que comienzan con `use` se llaman *Hooks*. `useState` es un Hook nativo dentro de React. Puedes encontrar otros Hooks nativos en la referencia de la API de React. También puedes escribir tus propios Hooks mediante la combinación de otros existentes.

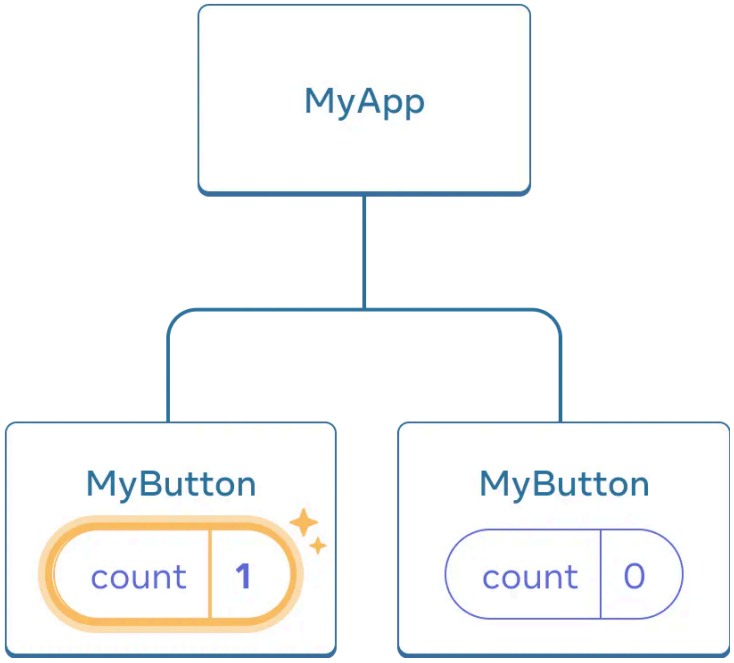
Los Hooks son más restrictivos que las funciones regulares. Solo puedes llamar a los Hooks *en el primer nivel* de tus componentes (u otros Hooks). Si quisieras utilizar `useState` en una condicional o en un bucle, extrae un nuevo componente y ponlo ahí.

Compartir datos entre componentes

En el ejemplo anterior, cada `MyButton` tenía su propio `count` independiente, y cuando hiciste clic en cada botón, solo el `count` del botón en hiciste clic cambiaba:



Inicialmente, cada estado `count` de `MyButton` es `0`.

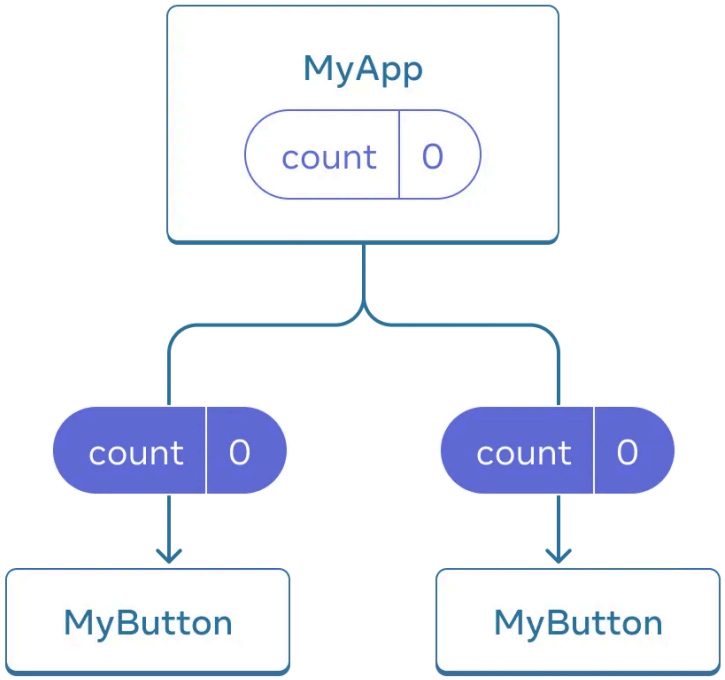


El primer `MyButton` actualiza su `count` a `1`.

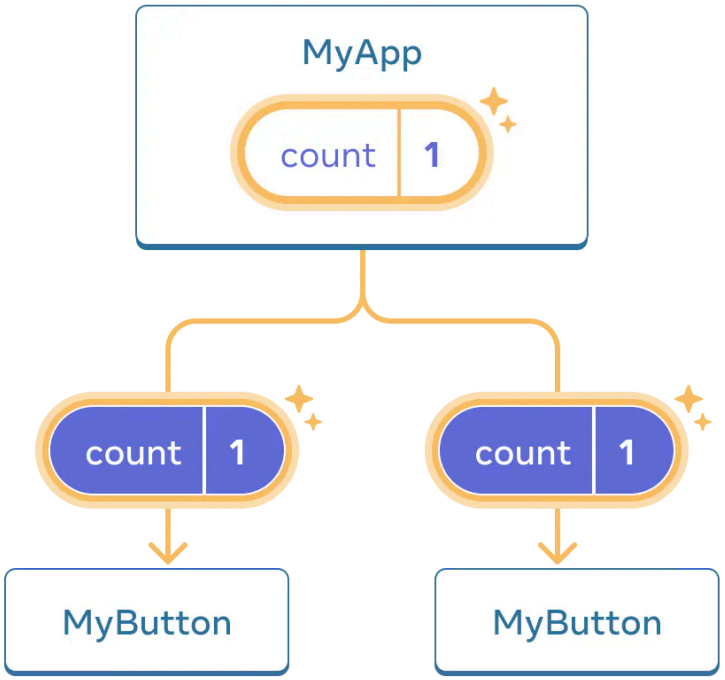
Sin embargo, a menudo necesitas que los componentes *compartan datos y se actualicen siempre en conjunto*.

Para hacer que ambos componentes `MyButton` muestren el mismo `count` y se actualicen juntos, necesitas mover el estado de los botones individuales «hacia arriba» al componente más cercano que los contiene a todos.

En este ejemplo, es `MyApp`:



Inicialmente, el estado `count` en `MyApp` es `0` y se pasa hacia abajo a los dos hijos.



Al hacer clic, `MyApp` actualiza su estado `count` a `1` y se lo pasa hacia abajo a ambos hijos.

Ahora cuando haces clic en cualquiera de los botones, `count` en `MyApp` cambiará, lo que causará que cambien ambos counts en `MyButton`. Aquí está como puedes expresarlo con código.

Primero, *mueve el estado hacia arriba* desde `MyButton` hacia `MyApp`:

```
export default function MyApp() {
  const [count, setCount] = useState(0);

  function handleClick() {
```

```
      setCount(count + 1);
    }

    return (
      <div>
        <h1>Contadores que se actualizan separadamente</h1>
        <MyButton />
        <MyButton />
      </div>
    );
}

function MyButton() {
    // ... estamos moviendo el código de aquí ...
}
```

Luego, *pasa el estado hacia abajo* desde `MyApp` hacia cada `MyButton`, junto con la función compartida para controlar el evento de clic. Puedes pasar la información a `MyButton` usando las llaves de JSX, de la misma forma como lo hiciste anteriormente con las etiquetas nativas `<img>`:

```
export default function MyApp() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
```

```
      <div>
        <h1>Contadores que se actualizan juntos</h1>
        <MyButton count={count} onClick={handleClick} />
        <MyButton count={count} onClick={handleClick} />
      </div>
    );
}
```

La información que pasas hacia abajo se llaman *props*. Ahora el componente `MyApp` contiene el estado `count` y el controlador de evento `handleClick`, y *pasa ambos hacia abajo como props* a cada uno de los botones.

Finalmente, cambia `MyButton` para que *lea* las props que le pasaste desde el componente padre:

```
function MyButton({ count, onClick }) {
  return (
    <button onClick={onClick}>
      Hiciste clic {count} veces
    </button>
  );
}
```

Cuando haces clic en el botón, el controlador `onClick` se dispara. A la prop `onClick` de cada botón se le asignó la función `handleClick` dentro de `MyApp`, de forma que el código dentro de ella se ejecuta. Ese código llama a `setCount(count + 1)`, que incremente la variable de estado `count`. El nuevo valor de `count` se pasa como prop a cada botón, y así todos muestran el nuevo valor.

Esto se llama «levantar el estado». Al mover el estado hacia arriba, lo compartimos entre componentes.

App.js

Descargar Reiniciar Bifurcar

```
import { useState } from 'react';

export default function MyApp() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <div>
      <h1>Contadores que se actualizan juntos</h1>
```

Mostrar más

Próximos pasos

¡En este punto ya conoces los elementos básicos de como escribir código en React!

Prueba el Tutorial para ponerlos en práctica y construir tu primera miniaplicación de React.

SIGUIENTE
Tutorial: Tres en línea ›

Comunidad

- Código de conducta
- Conoce al equipo
- Contribuidores de la documentación
- Agradecimientos

Más

- Blog
- React Native
- Privacidad
- Términos