useContext

> **Prueba la nueva documentación de React para `useContext`.**
>
> La nueva documentación reemplazará próximamente este sitio, que será archivado. Deja tu opinión aquí

```
const value = useContext(MyContext);
```

Acepta un objeto de contexto (el valor devuelto de `React.createContext`) y devuelve el valor de contexto actual. El valor actual del contexto es determinado por la propiedad `value` del `<MyContext.Provider>` ascendentemente más cercano en el árbol al componente que hace la llamada.

Cuando el `<MyContext.Provider>` ascendentemente más cercano en el árbol se actualiza, el Hook activa una renderización con el `value` más reciente del contexto pasado a ese proveedor `MyContext`. Incluso sí un ancestro utiliza `React.memo` o `shouldComponentUpdate`, una nueva renderización aún pasará empezando con el componente en si mismo usando `useContext`.

No olvides que el argumento enviado a `useContext` debe ser el *objeto del contexto en sí mismo*:

- **Correcto:** `useContext(MyContext)`
- **Incorrecto:** `useContext(MyContext.Consumer)`
- **Incorrecto:** `useContext(MyContext.Provider)`

Un componente que llama a `useContext` siempre se volverá a renderizar cuando el valor del contexto cambie. Si volver a renderizar el componente es costoso, puedes optimizar esto usando memorización.

> **Consejo**
>
> Si estás familiarizado con el API de contexto antes de Hooks, `useContext(MyContext)` es el equivalente a `static contextType = MyContext` en una clase, o a `<MyContext.Consumer>`.
>
> `useContext(MyContext)` solo te permite *leer* el contexto y suscribirte a sus cambios. Aún necesitas un `<MyContext.Provider>` arriba en el árbol para *proveer* el valor para este contexto.

Poniendo todo junto con Context.Provider

```
const themes = {
  light: {
    foreground: "#000000",
    background: "#eeeeee"
  },
  dark: {
    foreground: "#ffffff",
    background: "#222222"
  }
};

const ThemeContext = React.createContext(themes.light);

function App() {
  return (
    <ThemeContext.Provider value={themes.dark}>
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return (
    <button style={{ background: theme.background, color: theme.foreground }}>
      I am styled by theme context!
    </button>
  );
}
```