En el ejercicio de la tienda de Pokémon, hemos utilizado varios hooks de React para manejar el estado y la lógica de la aplicación. Los hooks que hemos usado son los siguientes:

useState:

```
Propósito: Es utilizado para manejar el estado en componentes funcionales.
Uso en el ejercicio:
Se utilizó para manejar el estado del carrito de compras (cart), permitiendo
agregar, actualizar y eliminar productos.
También se utilizó en PokemonList.js para almacenar la lista de Pokémones que
se obtienen de la API.

Ejemplo:

const [cart, setCart] = useState([]);
const [pokemons, setPokemons] = useState([]);
```

useEffect:

```
Propósito: Este hook se usa para manejar efectos secundarios en componentes
funcionales, como la obtención de datos de una API.
Uso en el ejercicio:
Se utilizó en PokemonList.js para hacer una llamada a la API de Pokémon cuando
el componente se monta y obtener la lista de Pokémones.

Ejemplo:
useEffect(() => {
fetch(`https://pokeapi.co/api/v2/pokemon?limit=10`)
    .then((response) => response.json())
    .then((data) => {
    const results = data.results;
    // Procesar los datos...
    setPokemons(pokemonData);
    })
    .catch((error) => console.error("Error fetching the pokemons: ", error));
}, []);
```

useContext:

```
Propósito: Este hook permite acceder a valores en un contexto, que es útil para
compartir datos o funciones a través de múltiples componentes sin necesidad de
pasar props manualmente.
Uso en el ejercicio:
Se utilizó para acceder al estado y funciones relacionadas con el carrito de
compras (addToCart, removeFromCart) en distintos componentes, como Product.js y
```

```
ShoppingCart.js.

Ejemplo:
const { addToCart } = useContext(CartContext);
```

Resumen
useState: Para manejar el estado local en los componentes, como la lista de Pokémones y el carrito de compras.
useEffect: Para manejar efectos secundarios, como la obtención de datos desde una API cuando un componente se monta.
useContext: Para acceder y compartir el estado global del carrito de compras a través de diferentes componentes.
Estos hooks permiten que la aplicación sea más eficiente y modular, aprovechando al máximo la funcionalidad de React.