Vamos a cubrir los siguientes temas en estos apuntes:

1. **Introducción a React**
2. **JSX y su importancia**
3. **Componentes funcionales y props**
4. **Estado y ciclo de vida (introducción breve)**
5. **Renderizado condicional**
6. **Creación del proyecto con Create React App**

📝 Apuntes de React: Nivel Básico

📌 1. Introducción a React

React es una biblioteca de JavaScript desarrollada por **Facebook** para construir interfaces de usuario (UI). A diferencia de otros frameworks como Angular o Vue, React se centra exclusivamente en la **capa de la vista** de la aplicación, es decir, la parte que los usuarios ven e interactúan. React permite construir interfaces dinámicas y eficientes mediante el uso de componentes reutilizables.

¿Por qué usar React?

- **Composición de componentes:** React permite dividir la interfaz en componentes independientes que se pueden reutilizar en diferentes partes de la aplicación.
- **Virtual DOM:** React utiliza un Virtual DOM para hacer actualizaciones eficientes en la interfaz. En lugar de actualizar todo el DOM, React solo actualiza los elementos que han cambiado.
- **Unidirectional Data Flow (Flujo de datos unidireccional):** Los datos fluyen en una sola dirección, lo que facilita la depuración y la gestión del estado.

📌 2. JSX: JavaScript + XML

JSX es una extensión de JavaScript que permite escribir código similar a HTML dentro de archivos JavaScript. Aunque JSX parece HTML, en realidad es una representación de código JavaScript. Es una de las características más distintivas de React y facilita la creación de interfaces de usuario.

Ejemplo básico de JSX:

```
const elemento = <h1>Hola, Mundo</h1>;
```

Este código es equivalente a:

```
const elemento = React.createElement('h1', null, 'Hola, Mundo');
```

¿Por qué usar JSX?

- **Más legible:** Permite ver la estructura de la interfaz de forma clara y concisa.
- **Facilita la creación de componentes complejos:** Con JSX, puedes combinar JavaScript y HTML, lo que facilita la creación de interfaces dinámicas e interactivas.

Reglas de JSX:

1. **Debe estar envuelto en un solo contenedor:** Puedes usar un `div`, un `section` o un fragmento (`<> </>`) como contenedor.
2. **Usar llaves para evaluar expresiones JavaScript:** Dentro de JSX, se pueden utilizar expresiones JavaScript usando llaves `{}`.

Ejemplo:

```
const nombre = "Pablo";
const saludo = <h1>Hola, {nombre}</h1>;
```

📌 3. Componentes Funcionales y Props

En React, los componentes son las piezas básicas que construyen la interfaz de usuario. Un componente puede ser una **función** o una **clase** (aunque las funciones son más comunes con los hooks modernos). Los componentes funcionales son simplemente funciones de JavaScript que retornan JSX.

Ejemplo de un componente funcional:

```
function Saludo() {
  return <h1>Hola, Mundo</h1>;
}
```

Para utilizar este componente, se debe renderizar dentro de otro componente o directamente en el archivo `index.js`:

```
<Saludo />
```

¿Qué son las props?

Las **props** (abreviatura de "properties") son parámetros que se pasan a los componentes para personalizar su contenido. Funcionan de manera similar a los parámetros de una función en JavaScript.

Ejemplo:

```
function Saludo(props) {
  return <h1>Hola, {props.nombre}</h1>;
}

// Uso del componente:
<Saludo nombre="Pablo" />
```

En este ejemplo, `props.nombre` recibe el valor `"Pablo"` y muestra el saludo personalizado.

📌 4. Estado y ciclo de vida (introducción breve)

El **estado** es una de las características más importantes de React y permite que los componentes respondan a cambios en los datos. A diferencia de las props, que son inmutables, el estado de un componente se puede actualizar, y al hacerlo, React vuelve a renderizar el componente para reflejar los cambios.

Uso básico del estado en un componente funcional:

Para manejar el estado en componentes funcionales, se usa el hook `useState`.

Ejemplo:

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

function Contador() {
  const [contador, setContador] = useState(0);

  return (
    <div>
      <p>Contador: {contador}</p>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}
```

En este ejemplo:

- `useState(0)` inicializa el estado `contador` con un valor de `0`.
- `setContador` es la función que se usa para actualizar el estado.
- Cada vez que se hace clic en el botón, el estado se actualiza y el componente se vuelve a renderizar.

📌 5. Renderizado condicional

En React, podemos mostrar contenido de forma condicional utilizando expresiones JavaScript dentro de JSX. Esto es útil para mostrar diferentes componentes o elementos en función del estado o las props.

Ejemplo:

```
function Mensaje({ esVisible }) {
  return (
    <div>
      {esVisible ? <p>El mensaje es visible</p> : <p>El mensaje no es visible</p>}
    </div>
  );
}
```

En este ejemplo:

- Utilizamos el operador ternario para decidir qué mensaje mostrar en función de la prop `esVisible`.

📌 6. Crear un proyecto con Create React App

Create React App es una herramienta que configura un proyecto de React con todo lo necesario para empezar a desarrollar, sin necesidad de configurar manualmente Webpack, Babel, o el entorno de desarrollo.

Comandos básicos:

1. Crear un proyecto:

   ```
   npx create-react-app mi-proyecto
   ```

2. Iniciar el servidor de desarrollo:

```
cd mi-proyecto
npm start
```

3. Crear un nuevo componente:
    - Crear un archivo `Saludo.js` dentro de `src/`.
    - Importarlo y usarlo en `App.js`.

Estructura inicial del proyecto:

```
src/
├── App.js
├── index.js
└── components/
    └── Saludo.js
```

Con estos apuntes, tienes una base sólida para entender React y trabajar en tus ejercicios. A medida que avances, puedes expandir estos temas con hooks más complejos, manejo del estado global, y el uso de librerías como React Router.

¡Espero que te sirvan! 🚀📖