S9. Ejercicios React + Javascript

ARRAYS

1. **Método map()**:
    - Crea una lista que muestre los nombres de una serie de amigos almacenados en un array utilizando el método `map()`.

```
const amigos = ["Ana", "Carlos", "Elena"];
// En el componente:
<ul>
  {amigos.map((amigo, index) => (
    <li key={index}>{amigo}</li>
  ))}
</ul>
```

1. **Método filter()**:
    - Crea una lista que muestre solo los números pares de un array utilizando el método `filter()`.

```
const numeros = [1, 2, 3, 4, 5];
// En el componente:
<ul>
  {numeros.filter(num => num % 2 === 0).map((num, index) => (
    <li key={index}>{num}</li>
  ))}
</ul>
```

1. **Método sort()**:
    - Ordena alfabéticamente un array de palabras y muéstralas en una lista utilizando el método `sort()`.

```
const palabras = ["Banana", "Apple", "Cherry"];
// En el componente:
<ul>
  {palabras.sort().map((palabra, index) => (
    <li key={index}>{palabra}</li>
```

```
  ))}
</ul>
```

1. **Método reduce()**:
    - Calcula la suma de un array de números utilizando el método `reduce()`.

```
const numeros = [1, 2, 3, 4, 5];
// En el componente:
<p>{numeros.reduce((acum, num) => acum + num, 0)}</p>
```

1. **Método find()**:
    - Encuentra el primer número mayor que 10 en un array utilizando el método `find()`.

```
const numeros = [5, 10, 15, 20];
// En el componente:
<p>{numeros.find(num => num > 10)}</p>
```

1. **Método every()**:
    - Verifica si todos los elementos de un array son mayores que 10 utilizando el método `every()`.

```
const numeros = [11, 12, 13, 14];
// En el componente:
<p>{numeros.every(num => num > 10) ? 'Todos son mayores que 10' : 'No todos son mayores qu
e 10'}</p>
```

1. **Método some()**:
    - Verifica si alguno de los elementos de un array es mayor que 10 utilizando el método `some()`.

```
const numeros = [5, 10, 15, 20];
// En el componente:
<p>{numeros.some(num => num > 10) ? 'Al menos uno es mayor que 10' : 'Ninguno es mayor que
10'}</p>
```

1. **Método indexOf()**:
    - Encuentra el índice del primer número 5 en un array utilizando el método `indexOf()`.

```
const numeros = [1, 2, 3, 4, 5];
// En el componente:
<p>{numeros.indexOf(5)}</p>
```

1. **Método findIndex()**:
    - Encuentra el índice del primer número mayor que 10 en un array utilizando el método `findIndex()`.

```
const numeros = [5, 10, 15, 20];
// En el componente:
<p>{numeros.findIndex(num => num > 10)}</p>
```

1. **Método join()**:
    - Junta todos los elementos de un array en una cadena de texto separados por comas utilizando el método `join()`.

```
const palabras = ["Hola", "Mundo"];
// En el componente:
<p>{palabras.join(', ')}</p>
```

Estos ejercicios te permitirán practicar diferentes métodos de arrays en React y entender cómo funcionan.

1. **Método slice()**:
    - Extrae y muestra los elementos de un array desde el índice 2 hasta el índice 5 utilizando el método `slice()`.

```
const numeros = [0, 1, 2, 3, 4, 5, 6];
// En el componente:
<ul>
  {numeros.slice(2, 5).map((num, index) => (
    <li key={index}>{num}</li>
```

```
  ))}
</ul>
```

1. **Método splice()**:
    - Elimina 2 elementos a partir del índice 3 de un array y muestra el array modificado utilizando el método `splice()`.

```
const numeros = [0, 1, 2, 3, 4, 5, 6];
// En el componente:
<ul>
  {numeros.splice(3, 2).map((num, index) => (
    <li key={index}>{num}</li>
  ))}
</ul>
```

1. **Método concat()**:
    - Combina dos arrays y muestra el resultado en una lista utilizando el método `concat()`.

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
// En el componente:
<ul>
  {arr1.concat(arr2).map((num, index) => (
    <li key={index}>{num}</li>
  ))}
</ul>
```

1. **Método fill()**:
    - Rellena un array con el número 0 y muestra el resultado utilizando el método `fill()`.

```
const numeros = [1, 2, 3, 4, 5];
// En el componente:
<ul>
  {numeros.fill(0).map((num, index) => (
    <li key={index}>{num}</li>
```

```
  ))}
</ul>
```

1. **Método reverse()**:
    - Invierte el orden de un array y muestra el resultado utilizando el método `reverse()`.

```
const numeros = [1, 2, 3, 4, 5];
// En el componente:
<ul>
  {numeros.reverse().map((num, index) => (
    <li key={index}>{num}</li>
  ))}
</ul>
```

1. **Método push() y pop()**:
    - Agrega un elemento al final de un array con `push()` y luego elimina un elemento del final con `pop()`, mostrando el resultado en una lista.

```
const numeros = [1, 2, 3, 4];
// En el componente:
// Agregar un elemento
numeros.push(5);
// Eliminar el último elemento
numeros.pop();
<ul>
  {numeros.map((num, index) => (
    <li key={index}>{num}</li>
  ))}
</ul>
```

1. **Método shift() y unshift()**:
    - Elimina el primer elemento de un array con `shift()` y luego agrega un elemento al inicio con `unshift()`, mostrando el resultado en una lista.

```
const numeros = [1, 2, 3, 4];
// En el componente:
// Eliminar el primer elemento
numeros.shift();
// Agregar un elemento al inicio
```

```
numeros.unshift(0);
<ul>
  {numeros.map((num, index) => (
    <li key={index}>{num}</li>
  ))}
</ul>
```

1. **Método flat()**:
    - Aplana un array de arrays y muestra el resultado en una lista utilizando el método `flat()`.

```
const arr = [[1, 2], [3, 4], [5, 6]];
// En el componente:
<ul>
  {arr.flat().map((num, index) => (
    <li key={index}>{num}</li>
  ))}
</ul>
```

1. **Método includes()**:
    - Verifica si un array incluye el número 5 utilizando el método `includes()`.

```
const numeros = [1, 2, 3, 4, 5];
// En el componente:
<p>{numeros.includes(5) ? 'Incluye el número 5' : 'No incluye el número 5'}</p>
```

1. **Método forEach()**:
    - Utiliza el método `forEach()` para sumar todos los elementos de un array y mostrar el resultado.

```
const numeros = [1, 2, 3, 4, 5];
let suma = 0;
// En el componente:
{numeros.forEach(num => suma += num)}
<p>{suma}</p>
```

OBJECTS

1. **Renderizar Propiedades**:
    - Crea un objeto con tus datos personales (nombre, edad, ciudad) y renderiza cada propiedad en un componente.

```
const persona = {
  nombre: "Juan",
  edad: 25,
  ciudad: "Madrid"
};
// En el componente:
<p>{persona.nombre}, {persona.edad}, {persona.ciudad}</p>
```

1. **Iterar sobre Propiedades**:
    - Crea un objeto y usa `Object.keys()` para iterar sobre sus propiedades y mostrarlas en una lista.

```
const objeto = { a: 1, b: 2, c: 3 };
// En el componente:
<ul>
  {Object.keys(objeto).map(key => (
    <li key={key}>{key}: {objeto[key]}</li>
  ))}
</ul>
```

1. **Agregar Propiedades**:
    - Agrega una nueva propiedad a un objeto y renderiza el objeto actualizado.

```
const objeto = { a: 1, b: 2 };
objeto.c = 3;
// En el componente:
<p>{JSON.stringify(objeto)}</p>
```

1. **Eliminar Propiedades**:
    - Elimina una propiedad de un objeto y renderiza el objeto actualizado.

```
const objeto = { a: 1, b: 2, c: 3 };
delete objeto.c;
// En el componente:
<p>{JSON.stringify(objeto)}</p>
```

1. **Acceder a Propiedades Anidadas**:
    - Crea un objeto con propiedades anidadas y accede a una propiedad anidada.

```
const objeto = { a: { b: { c: 3 } } };
// En el componente:
<p>{objeto.a.b.c}</p>
```

1. **Modificar Propiedades Anidadas**:
    - Modifica una propiedad anidada en un objeto y renderiza el objeto actualizado.

```
const objeto = { a: { b: { c: 3 } } };
objeto.a.b.c = 4;
// En el componente:
<p>{JSON.stringify(objeto)}</p>
```

1. **Objetos dentro de Arrays**:
    - Crea un array de objetos y renderiza una lista de una propiedad específica.

```
const personas = [{ nombre: "Ana" }, { nombre: "Carlos" }, { nombre: "Elena" }];
// En el componente:
<ul>
  {personas.map((persona, index) => (
    <li key={index}>{persona.nombre}</li>
  ))}
</ul>
```

1. **Filtrar Objetos**:
    - Crea un array de objetos y usa `filter()` para mostrar solo los objetos que cumplan una condición específica.

```
const personas = [{ nombre: "Ana", edad: 25 }, { nombre: "Carlos", edad: 30 }, { nombre:
"Elena", edad: 20 }];
// En el componente:
<ul>
  {personas.filter(persona => persona.edad > 25).map((persona, index) => (
    <li key={index}>{persona.nombre}</li>
  ))}
</ul>
```

1. **Ordenar Objetos**:
    - Crea un array de objetos y usa `sort()` para ordenarlos por una propiedad específica.

```
const personas = [{ nombre: "Ana", edad: 25 }, { nombre: "Carlos", edad: 30 }, { nombre:
"Elena", edad: 20 }];
// En el componente:
<ul>
  {personas.sort((a, b) => a.edad - b.edad).map((persona, index) => (
    <li key={index}>{persona.nombre}</li>
  ))}
</ul>
```

1. **Actualizar Propiedades**:
    - Crea un objeto y un botón que, al hacer clic, actualice una propiedad del objeto y muestre el objeto actualizado.

```
const [objeto, setObjeto] = React.useState({ a: 1 });

const actualizarObjeto = () => {
  setObjeto(prevObjeto => ({ ...prevObjeto, a: prevObjeto.a + 1 }));
};

// En el componente:
<button onClick={actualizarObjeto}>Actualizar</button>
<p>{JSON.stringify(objeto)}</p>
```

1. **Asignación de Propiedades**:
    - Utiliza `Object.assign()` para combinar dos objetos en uno nuevo y renderiza el resultado.

```
const objeto1 = { a: 1 };
const objeto2 = { b: 2 };
const objetoCombinado = Object.assign({}, objeto1, objeto2);
// En el componente:
<p>{JSON.stringify(objetoCombinado)}</p>
```

1. **Operador Spread**:
    - Utiliza el operador spread para combinar dos objetos en uno nuevo y renderiza el resultado.

```
const objeto1 = { a: 1 };
const objeto2 = { b: 2 };
const objetoCombinado = { ...objeto1, ...objeto2 };
// En el componente:
<p>{JSON.stringify(objetoCombinado)}</p>
```

1. **Comprobar Propiedades**:
    - Comprueba si un objeto tiene una propiedad específica utilizando `hasOwnProperty()`.

```
const objeto = { a: 1 };
// En el componente:
<p>{objeto.hasOwnProperty('a') ? 'Tiene la propiedad a' : 'No tiene la propiedad a'}</p>
```

1. **Objetos como Props**:
    - Pasa un objeto como prop a un componente y renderiza sus propiedades en el componente hijo.

```
const objeto = { a: 1, b: 2 };

const ComponenteHijo = (props) => (
  <p>{props.objeto.a}, {props.objeto.b}</p>
);

// En el componente:
<ComponenteHijo objeto={objeto} />
```

1. **Desestructuración de Objetos**:
    - Desestructura un objeto para acceder a sus propiedades más fácilmente.

```
const objeto = { a: 1, b: 2 };
const { a, b } = objeto;
// En el componente:
<p>{a}, {b}</p>
```

1. **Desestructuración en Parámetros**:
    - Desestructura un objeto directamente en los parámetros de una función.

```
const Componente = ({ a, b }) => (
  <p>{a}, {b}</p>
);
// En el componente:
<Componente {...{ a: 1, b: 2 }} />
```

1. **Comparar Objetos**:
    - Crea una función que compare dos objetos y determine si son iguales o no.

```
const compararObjetos = (obj1, obj2) => JSON.stringify(obj1) === JSON.stringify(obj2);
// En el componente:
<p>{compararObjetos({ a:

1 }, { a: 1 }) ? 'Son iguales' : 'Son diferentes'}</p>
```

1. **Objeto como Estado**:
    - Usa un objeto como estado en un componente y actualiza una propiedad del objeto.

```
const [objeto, setObjeto] = React.useState({ a: 1 });

const actualizarObjeto = () => {
  setObjeto(prevObjeto => ({ ...prevObjeto, a: prevObjeto.a + 1 }));
};

// En el componente:
```

```
<button onClick={actualizarObjeto}>Actualizar</button>
<p>{JSON.stringify(objeto)}</p>
```

1. **Objetos Constantes**:
    - Crea un objeto constante fuera de un componente y accede a sus propiedades en el componente.

```
const OBJETO_CONSTANTE = { a: 1, b: 2 };
// En el componente:
<p>{OBJETO_CONSTANTE.a}, {OBJETO_CONSTANTE.b}</p>
```

1. **Métodos de Objeto**:
    - Crea un objeto con un método y llama a ese método en un componente.

```
const objeto = {
  saludar: () => '¡Hola!'
};
// En el componente:
<p>{objeto.saludar()}</p>
```

1. **Método Object.values()**:
    - Crea un objeto y utiliza `Object.values()` para renderizar una lista con los valores del objeto.

```
const objeto = { a: 1, b: 2, c: 3 };
// En el componente:
<ul>
  {Object.values(objeto).map((valor, index) => (
    <li key={index}>{valor}</li>
  ))}
</ul>
```

1. **Método Object.entries()**:
    - Crea un objeto y utiliza `Object.entries()` para renderizar una lista con las entradas del objeto.

```
const objeto = { a: 1, b: 2, c: 3 };
// En el componente:
<ul>
  {Object.entries(objeto).map(([key, value], index) => (
    <li key={index}>{key}: {value}</li>
  ))}
</ul>
```

1. **Método Object.freeze()**:
    - Utiliza `Object.freeze()` para evitar que un objeto sea modificado, y verifica si el objeto es modificable o no.

```
const objeto = { a: 1 };
Object.freeze(objeto);
// Intenta modificar el objeto
objeto.a = 2;
// En el componente:
<p>{objeto.a}</p>
```

1. **Método Object.seal()**:
    - Utiliza `Object.seal()` para evitar que se agreguen o eliminen propiedades de un objeto, y verifica si puedes agregar una nueva propiedad o no.

```
const objeto = { a: 1 };
Object.seal(objeto);
// Intenta agregar una nueva propiedad
objeto.b = 2;
// En el componente:
<p>{'b' in objeto ? objeto.b : 'No se pudo agregar la propiedad b'}</p>
```