Emanuel Pontoni
https://github.com/epontoni

Ejercicios de React

1. ¿Qué son las props en un componente?
   Marque todas las opciones que apliquen.
   - **Son los parámetros del componente.**
   - Es el estado del componente.
   - Es el mecanismo para poner información dentro del componente.
2. ¿Qué es un componente?
   Marque una sola opción
   - Un elemento.
   - **Una función o clase que retorna un elemento.**
   - Un elemento que retorna una función o clase.
   - Otro:
3. ¿Cuáles de los siguientes tipos de componentes son válidos?
   Seleccione todas las opciones que apliquen.
   - Componente completo.
   - **Componente de clase.**
   - **Componente funcional.**
   - Componente padre.
   - Componente hijo.
4. ¿Cuál es el estado de un componente?
   El **estado** es un simple objeto JavaScript definido y administrado dentro del componente que nos sirve para representar el componente en un punto determinado en el tiempo, el estado (cuya definición no es obligatoria) se inicializa con un valor por defecto y puede ir mutando a lo largo de la vida del mismo (en contraste con las **props** son atributos de configuración para el componente y son recibidos desde un nivel superior (normalmente al instanciar el componente) y por definición son inmutables).
5. ¿Cuáles son los métodos del ciclo de vida de un componente?}
   Los métodos del ciclo de vida de un componente son funciones que pueden sobreescribirse para ejecutar código en distintos momentos particulares del proceso que dan lugar a la vida/actualización/muerte de un componente. Los métodos más usuales son:
   - El método `render()` es el único método requerido en un componente de clase.
   - El método constructor() se ejecuta antes de montar el componente.
   - El método `componentDidMount()` se ejecuta inmediatamente después que un componente se monte en el DOM.
   - El método `componentDidUpdate()` se invoca inmediatamente después de que la actualización ocurra.
   - El método `componentWillUnmount()` se invoca inmediatamente antes de desmontar y destruir un componente.

   Otros métodos no tan usuales son:
   - El método `shouldComponentUpdate()`.
   - `static getDerivedStateFromProps()`.
   - `getSnapshotBeforeUpdate()`.

Y hay otros que están obsoletos. La documentación completa se encuentra en: https://es.reactjs.org/docs/react-component.html#the-component-lifecycle.

6. ¿Cómo se establece el estado de un componente?
   El estado no se debe modificar directamente. Se debe realizar mediante la llamada al método `setState()`.
7. ¿Cuál de estas implementaciones cambia el estado de un componente? Elija una sola opción.
   - `this.state.value = "my value"`
   - **`this.setState({ value: "my value" })`**
   - `this.state = { "value": "my value" }`
   - Otro:_____
8. ¿Qué sucede cuando cambiamos el estado de un componente?
   El componente se vuelve a renderizar. Gracias a la invocación a `setState()`, React sabe que el estado cambió e invoca de nuevo al método `render()`.
9. Escriba un componente que renderice una lista de elementos dada una propiedad que contenga un arreglo de objetos de la siguiente forma: `{ firstname: "demo", lastname: "demo", dni: 1234 }`.

```
function ElementList(props) {
  const objects = props.objects;
  const listElements = objects.map((obj) =>
    <li key={obj.dni}>
      Firstname: {obj.firstname}, Lastname: {obj.lastname}, DNI:
{obj.dni}
    </li>
  );
  return (
    <ul>{listElements}</ul>
  );
}

const arrObjs = [
    {firstname: "demo1", lastname: "demo1", dni: 1111},
    {firstname: "demo2", lastname: "demo2", dni: 2222},
    {firstname: "demo3", lastname: "demo3", dni: 3333},
  ];

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render( <ElementList objects={arrObjs} />);
```

Clic aquí para ver el código en funcionamiento.

10. Escriba un componente que reciba una propiedad llamada “data” y renderice un H! con el contenido de data.

```
function RenderData(props) {
  const data = props.data;
  return <h1>{data}</h1>;
```

Emanuel Pontoni
https://github.com/epontoni

```
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render( <RenderData data={"The data"} />);
```

Clic aquí para ver el código en funcionamiento.

11. ¿Cómo se importan los módulos de JavaScript instalados con NPM o YARN? Marque sólo una opción.
    - `import name from 'node_modules/name';`
    - `import name from '../../node_modules/name/name.js'`
    - `import name from '../../node_modules/name/index.js'`
    - **`import name from 'name';`**
    - Others:_____
12. ¿Cómo puedo importar el siguiente componente, teniendo en cuenta que está dentro de un archivo en el mismo directorio, llamado `MyComponent.js`?

```
export default class MiComponente extends React.Component {
    render() {
        return (
            <h1>Hello world</h1>
        )
    }
}
```

    Marque sólo una opción
    - **`import MyComponent from './MyComponent';`**
    - `import { MyComponent } from 'MyComponent';`
    - `import MyComponent from 'MyComponent.js';`
    - `import { MyComponent } from './MyComponent.js';`
    - `Others: _____`
13. ¿Cuál es el error en el siguiente componente?

```
const MiComponente = () => {
    <h1>
        Hello World
    </h1>
}
```

    El error es que falta la instrucción `return`.

Emanuel Pontoni
https://github.com/epontoni

```
const MiComponente = () => {
  return <h1>
    Hola mundo
  </h1>
}
```

Si el cuerpo de la función flecha es una única declaración de retorno, podemos omitir la palabra clave return.

```
const MiComponente = () => <h1>Hola mundo</h1>
```

Si queremos separar en varias líneas la única expresión de retorno, encerramos entre paréntesis.

```
const MiComponente = () => (
  <h1>
    Hola mundo
  </h1>
)
```

14. Explique, con sus palabras, qué es lo que hace este componente.

```
export class Componente extends React.Component {


    constructor(props) {
        super(props);
        this.state = {
            property: 1
        }
    }

    click = (propertyValue) => {
        this.setState({ property: propertyValue + 1 })
    }

    render() {
        return (
            <>
                <h1>{this.state.property}</h1>
                <button onClick={() => this.click(this.state.property)}>
                    Click here
                </button>
            </>
        )
    }
}
```

Emanuel Pontoni
https://github.com/epontoni

El componente renderiza

- Un elemento **`h1`** con el valor de la propiedad "`property`" del estado (esta misma se establece en 1 cuando el componente es creado).
- Un elemento **`button`** con el texto "`Click here`" y el manejador de eventos `onClick` al cual se le pasa el método `click` con el valor de `property` como argumento y cuya finalidad es actualizar el estado de `property` incrementándose en una unidad mediante la llamada a `this.setState()`.

15. ¿Qué errores ves en el siguiente componente?

```
class Componente extends React.Component {
    render() {
        this.setState(property: 'Propiedad')
        return (
            <h1>El estado es: {this.state.property}</h1>
        )
    }
}
```

Errores:

- Falta el constructor de clase que asigne el `this.state` inicial.
- El argumento de `this.setState()` debe ser un objeto.
- La actualización del estado debe realizarse fuera del método `render()`, puesto que una actualización de estado provoca un re-renderizado del componente (entra así a un bucle infinito).

16. ¿Cómo se hace una llamada a la API Rest desde JavaScript?
Puede hacerse de tres formas diferentes: La manera más tradicional y/o "antigua" de llamar a una API es con **Ajax** (XMLHttpRequest), también se puede hacer con **fetch** o bien con la librería **axios**. Un ejemplo usando fetch (no soportado en IE)

```
fetch("https://swapi.dev/api/people/1/")
.then(res => {
  res.json().then(data => console.log(data))
})
.catch(err => {
  console.log('Algo salió mal...', err)
})
```

Click aquí para ver el código en funcionamiento (activar consola).

17. ¿Para qué sirve el hook useEffect?
Para ejecutar una función (efecto) después de que el componente se renderice.
18. ¿Cómo desarrollarías un hook personalizado que exponga métodos para guardar y modificar un contador? El hook debe tener un parámetro que permita al desarrollador especificar cuánto debe sumar el contador en cada llamada al setter.
Una posible implementación sería:

```
const useCounter = () => {
  const [counter, setCounter] = useState(0);
```

Emanuel Pontoni
https://github.com/epontoni

```
  const incrementar = (incremento = 1) => {
    setCounter(counter+incremento)
  }
  const reset = () => {
    setCounter(0)
  }
  return [
    counter,
    incrementar,
    reset
  ]
}
```

Click aquí para ver el código en funcionamiento.