React

Tema 2: Estado y Ciclo de Vida de los Componentes

Estado y Ciclo de Vida de los Componentes

- Props en ReactJS
- Estado en ReactJS
- Ciclo de vida de los componentes de clase
- Ejercicio práctico 1
- Ejercicio práctico 2

React

Props en ReactJS

React

Props en ReactJS

¿Qué son las props?

- Props es la abreviatura de "properties".
- Permiten pasar datos de un componente padre a un componente hijo.
- Facilitan la comunicación entre componentes.

Props en ReactJS

Uso de las props

- Las props son objetos que contienen valores o funciones.
- Se pasan como atributos a los componentes al crearlos.
- Pueden ser de cualquier tipo de dato

```
// Componente padre
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const message = 'Hola desde el componente padre';

  return <ChildComponent message={message} />;
}
```

```
// Componente hijo
import React from 'react';

const ChildComponent = (props) => {
  return <p>{props.message}</p>;
}
```

Props en ReactJS

Accediendo a las props

- Un componente hijo puede acceder a las props a través de su argumento de función.
- Las props son de solo lectura en el componente hijo.

```
// Componente padre
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const message = 'Hola desde el componente padre';

  return <ChildComponent message={message} />;
}
```

```
// Componente hijo
import React from 'react';

const ChildComponent = (props) => {
  return <p>{props.message}</p>;
}
```

Props en ReactJS

Inmutabilidad de las props

- Las props no deben ser modificadas directamente en el componente hijo.
- Solo el componente padre puede actualizar las props y enviar nuevos valores

Props en ReactJS

```
// Componente padre
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const name = 'John';

  return (
    <div>
      <ChildComponent name={name} />
    </div>
  );
}
```

```
// Componente hijo
import React from 'react';

const ChildComponent = (props) => {
  const { name } = props;

  const changeName = () => {
    // Intento de modificar directamente la prop name
    props.name = 'Jane'; // Esto generará un error en tiempo de
ejecución
  }

  return (
    <div>
      <p>Nombre: {name}</p>
      <button onClick={changeName}>Cambiar nombre</button>
    </div>
  );
}
```

Props en ReactJS

Beneficios de las props

- Comunicación estructurada entre componentes.
- Facilita la reutilización de componentes.
- Promueve la encapsulación y la modularidad del código.

Props en ReactJS

Resumen

- Las props permiten pasar datos de un componente padre a un componente hijo en React.
- Son de solo lectura en el componente hijo y deben mantenerse inmutables.
- Proporcionan una forma estructurada de comunicación entre componentes.

Props en ReactJS

Pasar datos de un componente padre a un componente hijo mediante props.

```
//Componente Padre:
import React from 'react';
import MiComponenteHijo from './MiComponenteHijo';

const MiComponentePadre = () => {
  const nombre = 'Juan';
  const edad = 25;

  return (
    <div>
      <h1>Componente Padre</h1>
      <MiComponenteHijo nombre={nombre} edad={edad} />
    </div>
  );
};
export default MiComponentePadre;
```

Props en ReactJS

Recepción y uso de props en el componente hijo:

```
//Componente Hijo:

import React from 'react';

const MiComponenteHijo = (props) => {
  return (
    <div>
      <h2>Componente Hijo</h2>
      <p>Nombre: {props.nombre}</p>
      <p>Edad: {props.edad}</p>
    </div>
  );
};

export default MiComponenteHijo;
```

Props en ReactJS

Propiedades por defecto

```
import React from 'react';

const MiComponente = (props) => {
  return (
    <div>
      <h1>Título: {props.titulo}</h1>
    </div>
  );
};

MiComponente.defaultProps = {
  titulo: 'Título por defecto'
};

export default MiComponente;
```

Puedes definir propiedades por defecto para un componente utilizando la propiedad **defaultProps**. Estas propiedades se utilizarán cuando no se pase un valor correspondiente para un prop determinado.

Props en ReactJS

Validación de props (Prop Types):

```
import React from 'react';
import PropTypes from 'prop-types';
const MiComponente = (props) => {
  return (
    <div>
      <h1>Título: {props.titulo}</h1>
    </div>
  );
};
MiComponente.propTypes = {
  titulo: PropTypes.string.isRequired
};

export default MiComponente;
```

npm install prop-types

Puedes realizar validaciones en los props utilizando la biblioteca prop-types. Esta biblioteca te permite definir reglas y tipos de datos para los props de un componente. Si los props no cumplen con las reglas definidas, se mostrará un mensaje de advertencia en la consola.

verificar tipos de datos, requerir propiedades, validar valores mínimos o máximos, etc.

Props en ReactJS

Acceso a los props dentro de un componente.

```
import React from 'react';

const MiComponente = (props) => {
  return (
    <div>
      <h1>Título: {props.titulo}</h1>
      <p>Contenido: {props.contenido}</p>
    </div>
  );
};

export default MiComponente;
```

React

Estado en ReactJS:

Estado en ReactJS:

Introducción al concepto de estado en React.

- Es una de las maneras en las que se procesan datos en esta librería de JavaScript
- El objeto props nos permite insertar datos estáticos
- Si necesitamos darle dinamismo a un elemento, (datos que se modifiquen a lo largo del tiempo), utilizaremos el estado en React.

Estado en ReactJS:

Introducción al concepto de estado en React.

- El estado en React, también conocido como state, es el segundo tipo de dato que maneja esta librería de JavaScript
- Un estado se compuesto por los datos internos que un componente puede manejar.
- Cada cambio de ese estado provocará que el elemento o componente se renderice de nuevo con una nueva representación en pantalla.

Estado en ReactJS:

¿Cómo utilizar el estado en React? Ejemplo 02_01

- Se inicia el documento HTML.

```
<!DOCTYPE html>
```

- Se especifica el título de la página y se agregan los scripts de React y ReactDOM desde una CDN. Estos scripts proporcionan las funcionalidades necesarias para ejecutar el código de React en el navegador.

```
<head>
  <title>Ejemplo de React con useState en HTML</title>
  <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
</head>
```

- Se crea un contenedor con el id "root". Este es el lugar donde se renderizará nuestro componente React.

```
<div id="root"></div>
```

Estado en ReactJS:

¿Cómo utilizar el estado en React? Ejemplo 02_01

- Se define el componente Example. Utilizamos la función React.useState para declarar una variable de estado llamada count con un valor inicial de 0, y la función setCount para actualizar dicho estado.

```
const [count, setCount] = React.useState(0);
```

- Se definen dos funciones, increment y decrement, que utilizan la función setCount para incrementar y decrementar el valor de count respectivamente.

```
const increment = () => {
      setCount(count + 1);
    };
const decrement = () => {
      setCount(count - 1);
    };
```

Estado en ReactJS:

¿Cómo utilizar el estado en React? Ejemplo 02_01

- Se utiliza React.createElement para construir la estructura del componente. Se crea un principal que contiene un <p> para mostrar el valor del contador (count), y dos botones <button> que llaman a las funciones increment y decrement respectivamente.

```
React.createElement('div', null,
        React.createElement('p', null, 'Contador: ' + count),
        React.createElement('button', { onClick: increment }, 'Incrementar'),
        React.createElement('button', { onClick: decrement }, 'Decrementar')
      )
```

- Se utiliza ReactDOM.render para renderizar el componente Example dentro del elemento con el id "root". Esto hace que el componente se muestre en el navegador.

```
ReactDOM.render(
    React.createElement(Example),
    document.getElementById('root')
  );
```

Estado en ReactJS:

Uso del estado para almacenar y manipular datos en un componente.

- El estado nos permite mantener información dinámica que puede cambiar durante la interacción del usuario con la aplicación.

- Ejemplo 02_02

Estado en ReactJS:

Ejemplo Ejemplo 02_02

- Se definió un componente de React llamado Example utilizando una función flecha (() => { ... }). Este componente utiliza React.useState para declarar una variable de estado llamada name y su función de actualización asociada setName. El valor inicial del estado es una cadena vacía ('').

```
const Example = () => {
const [name, setName] = React.useState('');
```

- Se definió una función handleChange que se ejecutará cada vez que se produzca un cambio en el input. Dentro de esta función, se utiliza setName para actualizar el estado name con el valor ingresado por el usuario (event.target.value).

```
const handleChange = (event) => {
    setName(event.target.value);
};
```

Estado en ReactJS:

Ejemplo Ejemplo 02_02

- Dentro del retorno del componente Example, se utiliza React.createElement para construir la estructura del componente. Se crea un principal que contiene un <p> para mostrar el valor del estado name, y un <input> vinculado al estado name a través del atributo value. Además, se configura el evento onChange del input para llamar a la función handleChange y actualizar el estado en respuesta a los cambios del usuario.

```
return (
      React.createElement('div', null,
        React.createElement('p', null, 'Nombre: ' + name),
        React.createElement('input', { type: 'text', value: name, onChange: handleChange })
      )
```

-

Estado en ReactJS:

Ejemplo Ejemplo 02_02

-Utilizando ReactDOM.render, se renderiza el componente Example dentro del elemento HTML con el id "root". Esto hace que el componente se muestre en el navegador.

```
ReactDOM.render(
  React.createElement(Example),
  document.getElementById('root')
);
```

Estado en ReactJS:

Inicializar el estado en un componente.

- Puedes inicializar el estado de un componente proporcionando un valor inicial al utilizar useState.
- Este valor inicial se utiliza para establecer el estado por primera vez cuando se renderiza el componente.

```
import React, { useState } from 'react';
const Example = () => {
  // Inicialización del estado con un valor inicial de 0
  const [count, setCount] = useState(0);

  // Resto del código del componente
  // ...
};
export default Example;
```

Estado en ReactJS:

Renderizar el componente en base a los cambios en el estado.

- Cuando el estado de un componente cambia, React se encarga automáticamente de volver a renderizar el componente para reflejar esos cambios en la interfaz de usuario.
- Esto se conoce como renderizado condicional, ya que el componente se renderiza de manera diferente según el estado actual.

- Ejemplo 02_03

React

Ciclo de vida de los componentes de clase

Ciclo de vida de los componentes de clase

componentDidMount: método invocado después de que el componente se haya montado en el DOM

- Este método se invoca automáticamente después de que el componente se haya montado en el DOM, es decir, cuando el componente ha sido creado y se ha adjuntado al árbol de elementos del DOM.
- se utiliza comúnmente para realizar **tareas de inicialización**, como la obtención de datos desde una API, establecer eventos o cualquier otra acción que deba realizarse una vez que el componente esté listo y visible en el DOM.

Ciclo de vida de los componentes de clase

componentDidUpdate: método invocado después de que el componente se haya actualizado en el DOM

Este método se invoca automáticamente después de que el componente se haya actualizado en el DOM, es decir, cuando se han realizado cambios en el componente y se han reflejado en la interfaz de usuario.

- se utiliza comúnmente para realizar acciones adicionales después de que el componente se haya actualizado, como realizar llamadas a API, actualizar el estado o interactuar con el DOM.

Ciclo de vida de los componentes de clase

componentWillUnmount: método invocado antes de que el componente se desmonte y sea eliminado del DOM.

Este método se invoca automáticamente antes de que el componente se desmonte y sea eliminado del DOM.

- se utiliza para realizar tareas de limpieza y liberación de recursos antes de que el componente se elimine, como cancelar suscripciones a eventos, detener temporizadores, eliminar escuchadores, entre otros..

React

Ejercicio

Ejercicio_1

Enunciado del ejercicio:

Crea un componente de React llamado PerfilUsuario que reciba las siguientes props: nombre, edad y foto. El componente debe renderizar la información del perfil de un usuario, incluyendo su nombre, edad y una imagen de perfil.

Además, el componente debe tener un botón que permita actualizar la edad del usuario cuando se hace clic.

Ejercicio_2

Enunciado del ejercicio:

Crea un componente de React llamado Calculadora que permita al usuario ingresar dos números. El componente debe calcular y mostrar la suma, resta, multiplicación y división de los dos números ingresados..

React

Test

Test

https://quizizz.com/admin/quiz/64ac4690b4d9e7001d0c9478?source=quiz_share