**Carrera:** Tecnología en Desarrollo de Software

**Nivel:** I I I Semestre

**Nombre del Estudiante:** Segura Pesantes Edson Gabriel

**Asignatura:** DISEÑO WEB

**Docente:** ING. ERNESTO ROBLES

2024 2025

## 1.- ¿Qué es un componente funcional en React y cómo se define?

Un componente funcional es una función de JavaScript que retorna JSX (una sintaxis similar a HTML) y describe cómo se debe ver una parte de la interfaz de usuario. Son más concisos, fáciles de leer y permiten el uso de Hooks, una herramienta poderosa para manejar estado y efectos secundarios.

```
function MiComponente() {
  return <h1>Hola desde un componente funcional</h1>;
}
```

## 2.- ¿Cuál es la diferencia entre un componente funcional y uno de clase?

| Aspecto | Funcional | De clase |
|---|---|---|
| **Definición** | Función de JS que retorna JSX | Clase que extiende React.Component |
| **Estado** | Usa useState (Hooks) | Usa this.state y this.setState |
| **Ciclo de vida** | Usa useEffect (Hooks) | Métodos como componentDidMount |
| **Codigo** | Más limpio y conciso | Más verboso |

### Ejemplo de clase:

```
class ComponenteClase extends React.Component {
  render() {
    return <h1>Hola desde una clase</h1>;
  }
}
```

### Ejemplo básico de un componente funcional:

```
function Saludo() {
  return <p>¡Hola mundo!</p>;
}
```

## 3.- ¿Qué son los props en React y cómo se pasan y reciben? ¿Para qué sirven?

Los props en React son datos que un componente padre le pasa a un componente hijo para que éste pueda mostrar contenido dinámico o comportarse de forma personalizada. Se pasan como atributos en la etiqueta del componente hijo y se reciben en el hijo como un objeto llamado props. Así, los props permiten que los componentes sean reutilizables y flexibles, ya que el hijo puede acceder a la información que le envía el padre y usarla dentro de su código.

## Muestra cómo pasar un prop desde un padre a un hijo

```
function Padre() {
  return <Hijo nombre="Carlos" />;
}

function Hijo({ nombre }) {
  return <p>Hola, {nombre}</p>;
}
```

## 4.- ¿Qué es el estado (state) en React?

Es un objeto que almacena datos propios y dinámicos de un componente, los cuales pueden cambiar con el tiempo. El estado permite que un componente sea interactivo y pueda actualizarse en respuesta a acciones del usuario, eventos o cambios internos, haciendo que la interfaz gráfica se renderice de nuevo con la información actualizada. A diferencia de los props, el estado es local **y** mutable dentro del componente.

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

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

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

## 5.- ¿Qué es el ciclo de vida de un componente en React?

Son las etapas que atraviesa un componente desde que se crea hasta que se destruye. Estas etapas incluyen el montaje (cuando el componente se inserta en la interfaz), la actualización (cuando cambia su estado o props y se vuelve a renderizar) y el desmontaje (cuando se elimina de la interfaz). Entender el ciclo de vida permite ejecutar acciones específicas en cada etapa, como cargar datos, actualizar información o limpiar recursos.

```
import { useEffect } from 'react';

useEffect(() => {
  console.log("Componente montado");
  return () => {
    console.log("Componente desmontado");
  };
}, []);
```

## 6.- ¿Qué es JSX?, ¿Qué es y cómo se usa en React?

Es una sintaxis que permite escribir código similar a HTML dentro de JavaScript. En React, se usa para definir la estructura visual de los componentes de forma clara y legible, combinando la lógica y la presentación en un mismo lugar. JSX se transpila a llamadas a funciones de React que crean los elementos que se renderizan en la interfaz.

```
function Usuario({ nombre, edad }) {
  return (
    <div>
      <h2>Hola, {nombre}!</h2>
      <p>Tienes {edad} años.</p>
      {edad >= 18 ? <p>Eres mayor de edad.</p> : <p>Eres menor de edad.</p>}
    </div>
  );
}
```

## 7.- ¿Qué problemas intenta resolver React con los Hooks?

React introdujo los Hooks para resolver problemas como la dificultad de reutilizar lógica entre componentes, la complejidad y verbosidad del código en componentes de clase, y la necesidad de usar clases para manejar estado y ciclo de vida. Los Hooks permiten usar estado y otras funcionalidades de React en componentes funcionales, haciendo el código más simple, claro y fácil de mantener.

### Ejemplo de uso de useState, useMemo, useEffect

```jsx
import { useState, useEffect, useMemo } from 'react';

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

  useEffect(() => {
    console.log("Se actualizó el contador: ", contador);
  }, [contador]);

  const numeroPesado = useMemo(() => {
    console.log("Calculando valor costoso...");
    return contador * 1000;
  }, [contador]);

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

# Bibliografía


- React Team. (s.f.). *Getting Started with React*. React Official Documentation. Recuperado de https://reactjs.org/docs/getting-started.html

- React Team. (s.f.). *Introducing Hooks*. React Official Documentation. Recuperado de https://reactjs.org/docs/hooks-intro.html

- Stefanov, S. (2016). *React – Up & Running*. O'Reilly Media.

- Chinnathambi, K. (2018). *Learning React: Functional Web Development with React and Redux*. Addison-Wesley Professional.

- Mozilla Contributors. (s.f.). *React — Getting Started*. MDN Web Docs. Recuperado de https://developer.mozilla.org/es/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started