

# JavaScript Avanzado

## Sesión 12



Universidad  
Tecnológica  
del Perú

# ¿Tienen alguna consulta o duda sobre la clase previa?



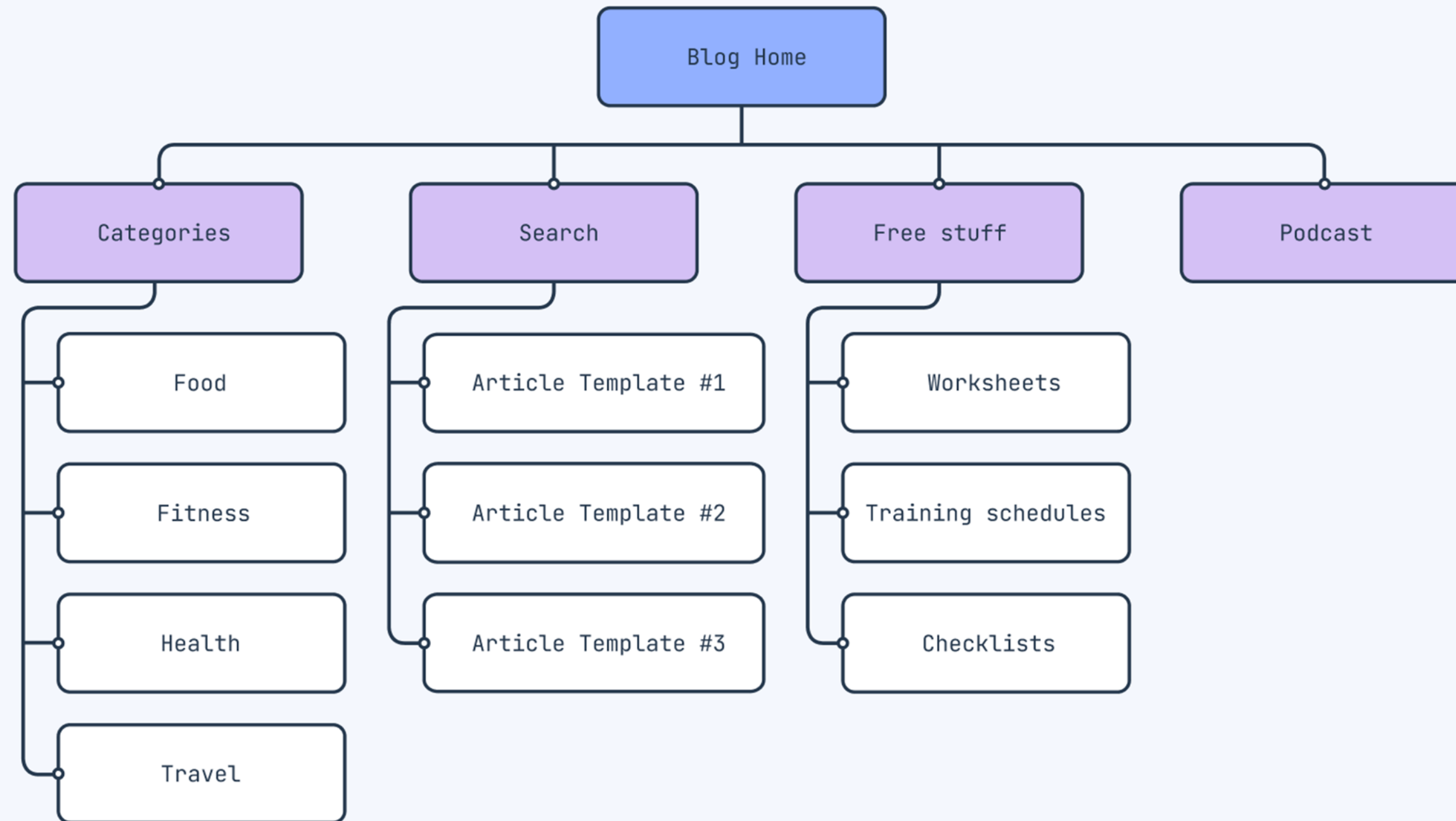
# Logro de la sesión

Al finalizar la sesión, el estudiante construye una aplicación web empleando la biblioteca React mediante componentes, un sistema de navegación y el procesamiento de formularios.

# ¿Cómo implementar la navegación con React?



# Website structure



*¿Cuál es la importancia de un manejador de rutas en los proyectos web?*



# Contenido

## Elementos de React

- Navegación
- Componentes
- Formularios
- Implementación de una aplicación web mediante React.





# Navegación

- Para implementar la navegación en una aplicación React utilizando TypeScript (TSX), se debe instalar React Router.
- Dentro del proyecto ya creado se debe ejecutar (con **node.js**) el comando de instalación de **React Router DOM**.

```
npm install react-router-dom
```



```

1  import * as React from "react";
2  import { createRoot } from "react-dom/client";
3  import {
4    createBrowserRouter,
5    RouterProvider,
6    Route,
7    Link,
8  } from "react-router-dom";
9
10 const router = createBrowserRouter([
11   {
12     path: "/",
13     element: (
14       <div>
15         <h1>Hello World</h1>
16         <Link to="about">About Us</Link>
17       </div>
18     ),
19   },
20   {
21     path: "about",
22     element: <div>About</div>,
23   },
24 ]);
25
26 createRoot(document.getElementById("root")).render(
27   <RouterProvider router={router} />
28 );

```



```

1  import { StrictMode } from 'react'
2  import { createRoot } from 'react-dom/client'
3  import './index.css'
4  import { RouterProvider } from 'react-router-dom'
5  import router from './routes/routes'
6
7  createRoot(document.getElementById('root')).render(
8    <StrictMode>
9      <RouterProvider router={router} />
10    </StrictMode>,
11  )

```

```

const router = createBrowserRouter(
  [
    {
      path: "/",
      element: <App />,
      errorElement: <><h2>Error</h2><a href="/">Regresar al inicio</a></>,
      children: [
        {
          errorElement: <h2>Error</h2>,
          children: [
            { index: true, element: <Inicio />, },
            { path: "inicio", element: <Inicio /> },
            { path: "contacto", element: <Contacto /> },
            { path: "nosotros", element: <Nosotros /> },
            { path: "productos", element: <Productos /> },
            { path: "iniciar-sesion", element: <InicioSesion /> },
            { path: "*", element: <ErrorPage /> },
          ]
        }
      ]
    }
  ]
);

```



```
const App: FC = () => {
  return (
    <>
      <HeaderComponent />
      <main>
        <hr />
        <Outlet />
        <hr />
      </main>
      <FooterComponent />
    </>
  )
}
```



```
const HeaderComponent: FC = () => {
  return (
    <header>
      <Navegador />
    </header>
  );
};
```



```
const Navegador: FC = () => {
  return (<>
    <nav className={`menu`}>
      <Link to={`inicio`}>
        <div className={`brand`}>
          <img src={vite} style={{ width: "1rem", height: "auto" }} />
          <span>Michi Store</span>
        </div>
      </Link>
      <ul>
        <li><Link to={`nosotros`}>Nosotros</Link></li>
        <li><Link to={`productos`}>Productos</Link></li>
        <li><Link to={`contacto`}>Contacto</Link></li>
      </ul>
      <div className={`icons`}>
        <Link className={`icon-button`} to={`carrito`}>
          <span>C</span><span>Carrito</span>
        </Link>
        <Link className={`icon-button`} to={`iniciar-sesion`}>
          <span>I</span><span>Mi cuenta</span>
        </Link>
      </div>
    </nav>
  </>);
}
```

# Componentes

- Un componente en React es básicamente una pieza reutilizable de la **interfaz de usuario**.
- Es como un bloque de construcción que encapsula su propia estructura (**HTML**), estilos (**CSS**) y comportamiento (**JavaScript**).
- Permite construir interfaces complejas dividiéndolas en piezas más pequeñas y manejables.
- **Por ejemplo**, se puede tener un componente para un botón, otro para un encabezado, y otro para una tarjeta de perfil. Luego combinar estos componentes en componentes más grandes para construir una aplicación.

# Componentes integrados

- **Fragment**, escrito alternativamente como `<>...</>`, permite agrupar varios nodos JSX o TSX juntos.
- **Profiler** permite medir el rendimiento de la renderización de un árbol de **React** de manera programática.
- **Suspense** te permite mostrar un sustituto mientras los componentes hijos se están cargando.
- **StrictMode** permite controles adicionales sólo para desarrollo que ayudan a encontrar errores anticipadamente.

# Componentes de clase

- **Sintaxis**: Se crean usando la sintaxis de clases de ES6, extendiendo **React.Component**.
- **Estado**: El estado se define dentro de la clase usando **this.state** y se actualiza con **this.setState()**.
- **Ciclo de vida**: Tienen métodos de ciclo de vida como **componentDidMount**, **componentDidUpdate** y **componentWillUnmount**.
- Usan **this** para acceder a las propiedades y métodos de la clase.



# Componentes funcionales

- **Sintaxis:** Se crean usando funciones de JavaScript.
- **Estado y efectos:** Usan **hooks** como **useState** y **useEffect** para manejar el estado y los efectos secundarios.
- **Ciclo de vida:** No tienen métodos de ciclo de vida, pero se pueden usar **hooks** para lograr un comportamiento similar.
- No usan **this**, ya que son funciones simples.



## Componente de clase

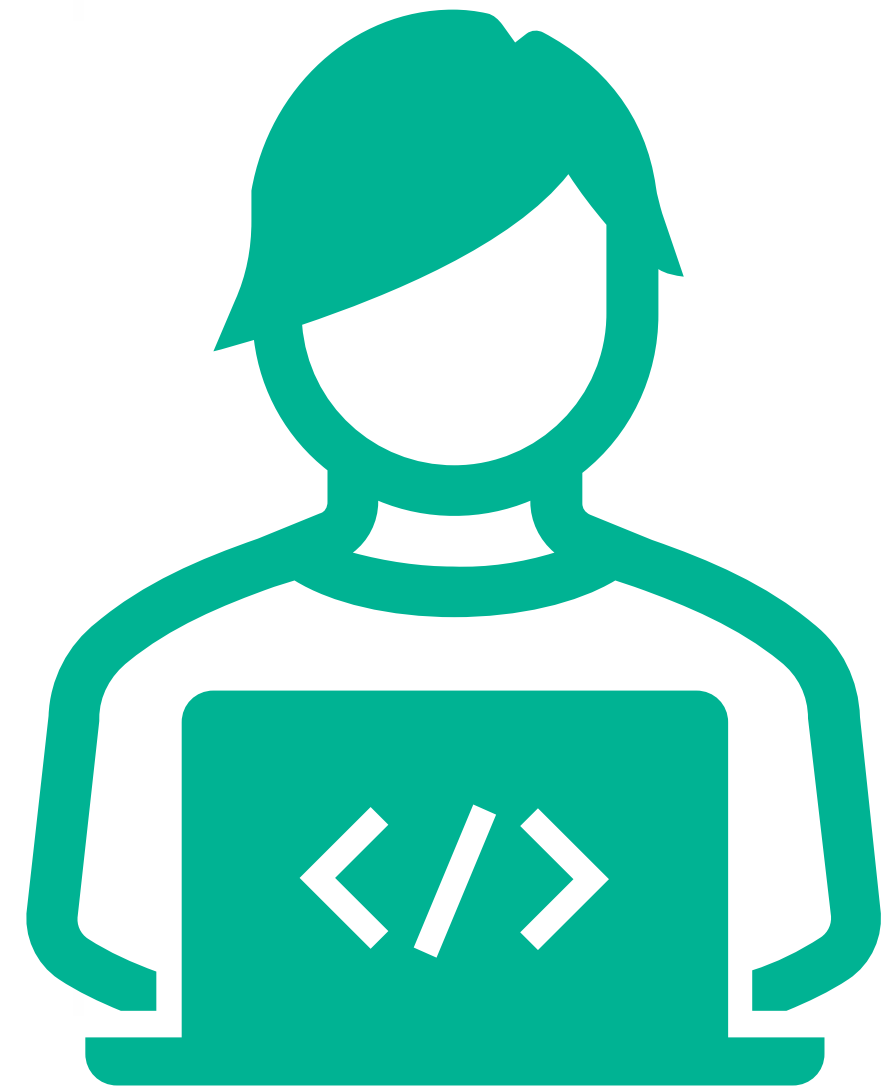
```
class ComponenteClase extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { contador: 0 };  
  }  
  
  componentDidMount() {  
    // Código que se ejecuta después de que el componente se monta.  
  }  
  
  render() {  
    return <div>{this.state.contador}</div>;  
  }  
}
```

```
function ComponenteFuncional() {  
  const [contador, setContador] = useState(0);  
  
  useEffect(() => {  
    // Código que se ejecuta después de que el componente se monta.  
  }, []);  
  
  return <div>{contador}</div>;  
}
```

## Componente funcional

# Programando


- Construir un menu de navegación con **react-router-dom** para navegar entre distintas páginas construidas mediante **componentes**.



# Formularios en React

- Los formularios en React se pueden procesar de diversas formas, desde una manipulación con JS básica, el uso de bibliotecas de terceros o mediante sus Hooks
- Tenemos una clasificación dependiendo del control:
  - **Formularios con control:** React gestiona el estado de los campos del formulario. Cada cambio en un campo de entrada actualiza el estado de un componente mediante el uso de `useState` u otro mecanismo de estado.
  - **Formularios sin control:** los valores de los campos de entrada no se gestionan directamente con el estado de **React**. En lugar de eso, se accede a los valores de los inputs directamente desde el DOM mediante referencias

# Procesar el “submit”



```
const handleSubmit = (e: FormEvent) => {  
  e.preventDefault(); // Evita el comportamiento por defecto del formulario  
  const formData = new FormData(e.target as HTMLFormElement);  
  const payload = Object.fromEntries(formData);  
  // Aquí puedes enviar la información a una API o procesarla como necesites  
  console.log('Datos enviados:', payload);  
};
```

```
return (<>  
  <h3>Contacto</h3>  
  <form onSubmit={handleSubmit} className="contacto">  
    <label htmlFor="nombre">Nombre</label>  
    <input type="text" name="nombre" id="nombre" required />  
    <label htmlFor="correo">Correo</label>  
    <input type="email" name="correo" id="correo" required />  
    <label htmlFor="mensaje">Mensaje</label>  
    <textarea id="mensaje" name="mensaje" required></textarea>  
    <button type="submit">Enviar</button>  
  </form>  
</>);
```

# Formularios sin estado “useRef”



```
return (<>
  <h3>Suscribirse</h3>
  <form onSubmit={handleSubmitUncontrolled} className="contacto">
    <label htmlFor="mail">Correo</label>
    <input type="email" id="mail" ref={inputRef} />
    <button type="submit">Suscribirse</button>
  </form>
</>);
```

```
const inputRef: any = useRef(null);

const handleSubmitUncontrolled = (e: FormEvent) => {
  e.preventDefault();
  console.log('Valor enviado:', inputRef.current.value);
};
```



# Formularios con estado “useState”

```
const handleLoginSubmit = (e: FormEvent) => {
  e.preventDefault();
  console.log('Valores enviados:', correo, clave);
  //Limpiar
  setClave('');
  setCorreo('');
};

return (<>
  <h3>Iniciar sesión</h3>
  <form className="login" onSubmit={handleLoginSubmit}>
    <label htmlFor="correo">Correo</label>
    <input type="email" name="correo" id="correo" required
      value={correo} onChange={handleCorreoChange} />
    <label htmlFor="clave">Contraseña</label>
    <input type="password" name="clave" id="clave" required
      value={clave} onChange={handleClaveChange} />
    <button type="submit">Ingresar</button>
  </form>
</>);
```

```
const [correo, setCorreo] = useState("");
const [clave, setClave] = useState("");

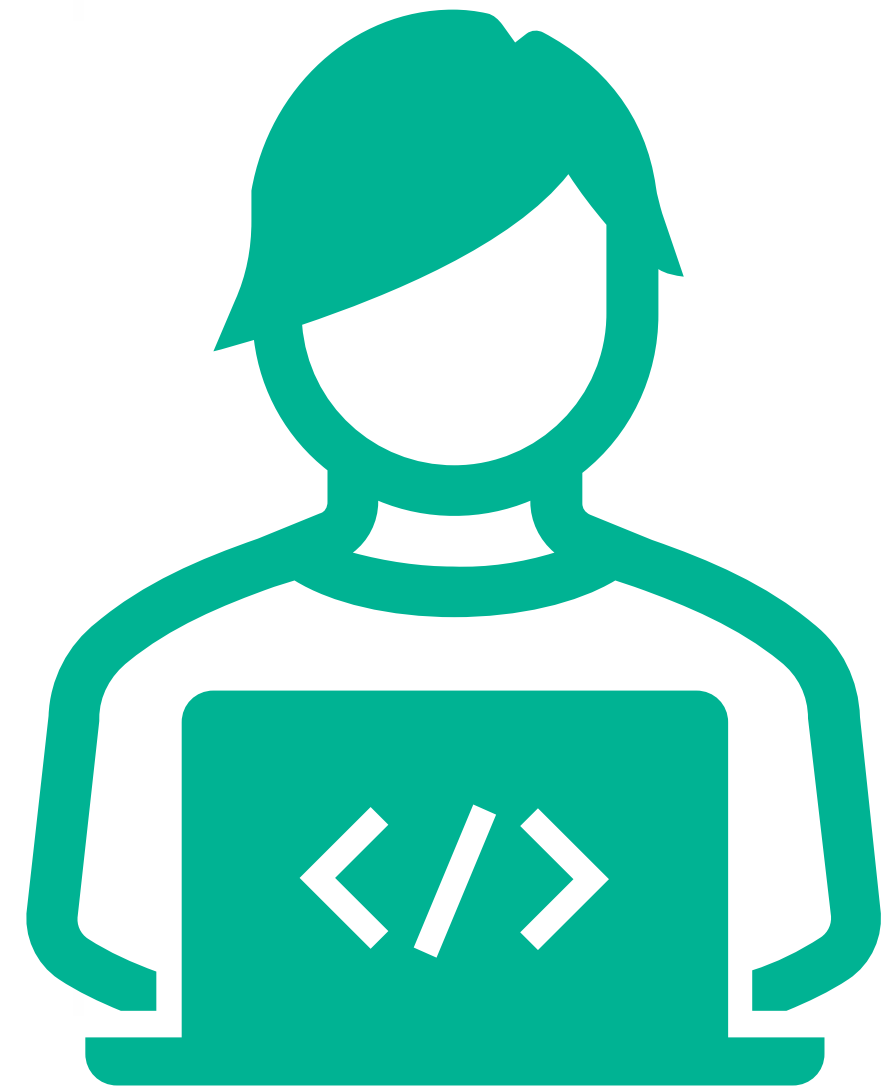
const handleCorreoChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  // Actualiza el estado con el nuevo valor
  setCorreo(e.target.value);
};

const handleClaveChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  // Actualiza el estado con el nuevo valor
  setClave(e.target.value);
};
```



# Programando

- Procesamos formularios empleando React.



# ¿Tienen alguna consulta o duda?



# Actividad



Resolver la actividad planteada en la plataforma.

# Cierre

## ¿Qué hemos aprendido hoy?



Elaboramos nuestras conclusiones sobre el tema tratado





**Universidad  
Tecnológica  
del Perú**