Actividad de Aprendizaje N° 08A

JSX y Componentes en React

Compiled by: Mg. Jaime Suasnábar Terrel

1. Renderizar HTML en React

La función de renderizado

Modifica el DOM, con el código HTML de un componente sobre un elemento html con un id generalmente llamado root o main.

Sintaxis:

```
ReactDOM.render(<name_component />, element)
```

La función ReactDOM.render() tiene dos argumentos:

1. Código HTML
2. Elemento HTML a ser modificado.

¿Renderizar qué, cómo y dónde?

```
<body>

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

    <script type="text/babel">

        function saludo() {
            let txt = " por Jaime";
            return <h1>Hola Mundo! {txt} </h1>;
        }

        ReactDOM.render(<saludo />, document.getElementById('root'))

    </script>
</body>
```

¿Cómo funciona Renderizar en proyecto vite?

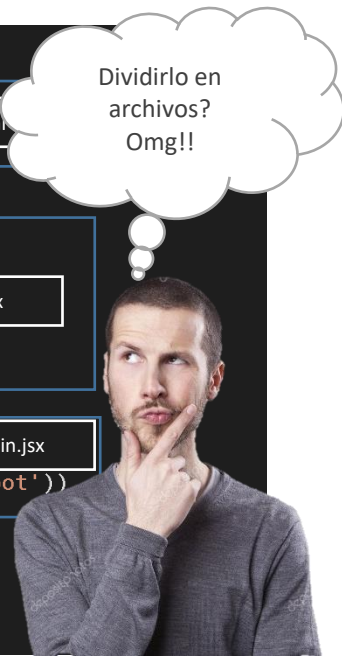
El código se lleva a diferentes archivos en React actualmente

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

    <script type="text/babel">
        function saludo() {
            let txt = " por Jaime";
            return <h1>Hola Mundo! {txt} </h1>;
        }

        ReactDOM.render(<saludo />, document.getElementById('root'))

    </script>
</body>
```

Index.html

App.jsx

Main.jsx



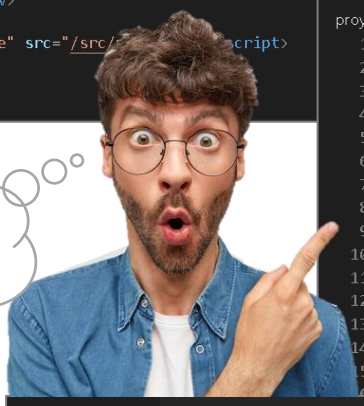
Quedando así:

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/src/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite App</title>
  </head>
  <body>
    <div id="root"></div>

    <script type="module" src="/src/
  </body>
</html>
```

App.jsx

```
import { useState } from 'react'
import logo from './logo.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <div className="App">

    </div>
  )
}

export default App
```



Main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
)
```

2. JSX y React

¿Qué es JSX?

JSX es *JavaScript Sintax Extension*, es decir una mezcla de *JS y HTML* fue creada por *Facebook* para el uso con su *librería React*. JSX requiere transformarse a JavaScript mediante un transpilador como Babel.

Con JSX y React puedes crear código reutilizable y estructurar fácilmente tu aplicación desde una mentalidad basada en componentes.

Variable o Constante

Las variables son como en JavaScript, pero se puede guardar código html.

Similar a JS

```
const myId = 'test'
const numSide = 6
```

Así es como se define una etiqueta <h1> que contiene una cadena:

```
const element = <h1>Hola, Mundo! </h1>
```

¿Lleva comilas? NO

Parece una extraña mezcla de JavaScript y HTML, pero en realidad es todo JavaScript.

Expresiones en JSX

Con JSX puedes escribir expresiones dentro de llaves { }. Estos son ejecutados como JS.

Dentro de una expresión JSX, con el operador { } se pueden:

- Insertar las variables
- Insertar atributos de html
- Ejecutar código JS

Insertar variables

```
const usuario = 'Jaime'
const element = <h1>Bienvenido {usuario}!!!  </h1>
```

Insertar atributos

```
const usuario = 'Jaime'
const myId = 'titulo'
const element = <h1 id={myId}>Bienvenido {usuario}!!!  </h1>
```

Ejecutar JS

```
const usuario = 'Jaime'
const myId = 'titulo'
const element = <h1 id={myId}>Bienvenido {usuario}!!! {2000 + 22} </h1>
```

Se agregó: {2000 + 22} que devuelve la suma //2022.

Insertar un bloque grande de HTM

Para escribir HTML en varias líneas, coloque el HTML entre paréntesis:
Ejemplo 01.

```
const myElement = (
  <ul>
    <li>Huancayo</li>
    <li>Arequipa</li>
    <li>Trujillo</li>
  </ul>
);
```

Un elemento de nivel superior

El código HTML debe estar *envuelto en:*

- *UN elemento de nivel superior*
- Un fragmento.

Entonces, si desea escribir dos párrafos, debe colocarlos dentro de un elemento principal, como un .
Ejemplo: Envuelva dos párrafos dentro de un elemento DIV:

```
const myElement = (
  <div>
    <p>Primer parrafo.</p>
    <p>Segundo parrafo.</p>
  </div>
);
```

O Alternativamente, puede usar un "fragmento" para envolver varias líneas. Esto evitará agregar innecesariamente nodos adicionales al DOM.
Un fragmento parece una etiqueta HTML vacía: <></>.
Ejemplo: Envuelve dos párrafos dentro de un fragmento:

```
const myElement = (
  <>
    <p>Primer parrafo.</p>
    <p>Segundo parrafo.</p>
  </>
);
```

Los elementos deben estar cerrados

JSX sigue las reglas XML y, por lo tanto, los elementos HTML deben tener su etiqueta de apertura y cierre. Algunas etiquetas no cierran, utilice “/>”.

Ejemplo
Cierra los elementos vacíos con/>

```
const myElement = <input type="text" />;
```

Clase de atributo = className

El atributo **class** es un atributo muy utilizado en HTML, pero dado que JSX se representa como JavaScript y la palabra clave ***class*** es una palabra reservada en JavaScript, no puede utilizarla en JSX.

Utilice el atributo **`className`** en su lugar.

Ejemplo: Utilice el atributo className en lugar de class en JSX:

```
const myElement = <h1 className="myclass">Hola MUndo</h1>;
```

Condicional IF

React permite la sentencia if, pero no dentro del HTML JSX.

Para usar los IF en JSX, debe colocarlo fuera de HTML JSX, o podría usar una expresión ternaria en su lugar:

Ejemplo: Escribe "Menor de Edad" si nEdad es menor a 10, de lo contrario "Mayor de Edad" con un if:

```
const nEdad = 5;
let text = "Mayor de Edad";
if (x < 10) {
  text = "Menor de Edad";
}

const myElement = <h1>{text}</h1>;
```

Ejemplo: Escribe "Menor de Edad" si nEdad es menor a 10, de lo contrario "Mayor de Edad" con un operador ternario:

```
const x = 5;

const myElement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>;
```

Recomendaciones de Codificación en JSX

- JSX nos permite escribir elementos HTML en JavaScript y colocarlos en el DOM sin estos métodos: createElement() o .appendChild().
- JSX convierte etiquetas HTML en elementos reactivos.
- El index.html llama al main.jsx, es decir es el primer archivo en ser ejecutado.
- No es necesario que use JSX, pero JSX facilita la escritura de aplicaciones React.

Ejemplo 01.

Refactorice main.jsx para mostrar un mensaje en <h1> de la FIS UNCP con JSX.

```
const myElement = <h1>I Love FIS</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Ejemplo 02.

Refactorice main.jsx para mostrar un mensaje en <h1> de la FIS UNCP sin JSX.

```
const myElement = React.createElement('h1', {}, 'Yo amo a la FIS!');

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Aquí se mostró dos ejemplos. El primero usa JSX y el segundo no:

Ejemplos

3. Componentes en React

¿Qué es un Componente?

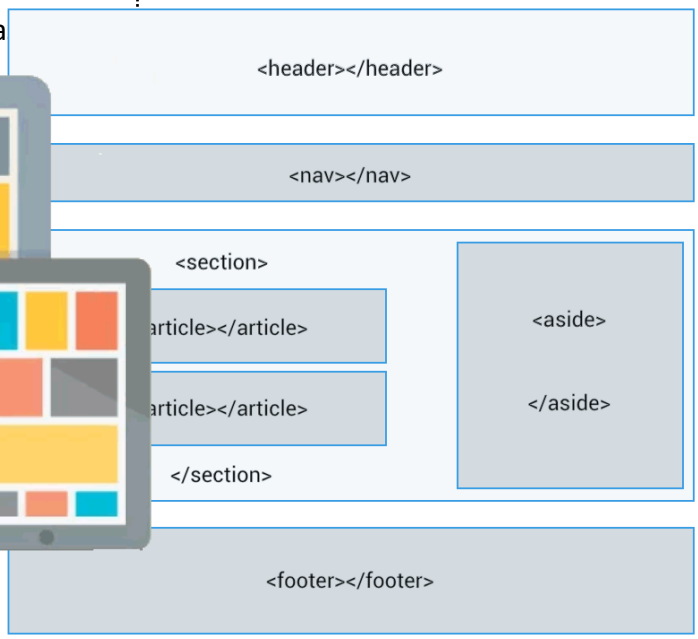
Un *componente de software* es una unidad modular con interfaces y dependencias bien definidas que permiten *ofrecer y/o solicitar servicios o funcionalidades*.

¿Qué es la Programación basada en Componentes?

Es una rama de la ***ingeniería del software***, con énfasis en la **descomposición de sistemas** en ***componentes funcionales o lógicos*** con interfaces bien definidas usadas para la comunicación entre componentes.

Web Components

En las *páginas web,* una buena práctica es dividirlo en *componentes* tomando en cuenta el html sémático, ya que muchos de estos componentes serán reutilizada en diversas página



Asimismo, cada componente debe contar con sus estilos para cada tipo de dispositivo, es decir respetando el diseño responsivo.

Component Layout

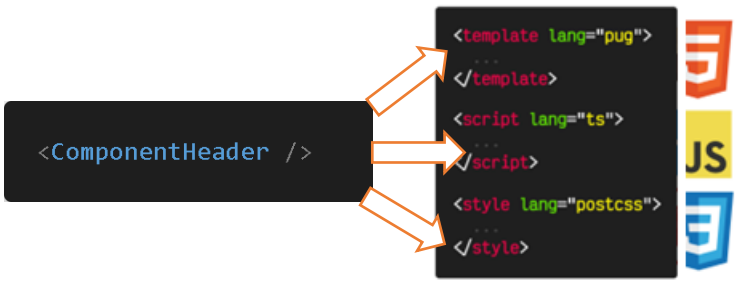
Un componente es código independiente y reutilizable representado por un nombre
Sintaxis:

```
<Nombre-componente />
o
<NombreComponente />
```

Tienen el mismo propósito que una función, pero funcionan de forma aislada y devuelven HTML.

El nombre de un componente en React debe empezar con una letra Mayúscula.
Todo componente se subdivide en 3 partes:

1. Código html
2. Código JavaSc
3. Código CSS




```
function App() {
  //PARTE DEL JS

  return (
    //PARTE DEL HTML Y CSS
    <div className="App">

    </div>
  )
}

export default App
```

Tipos de Componentes en React

Son de dos tipos:

- Componentes de Clase
- Componentes de Función

Componente de clase

Un componente de clase debe incluir la palabra reservada extends React.Component. Esta crea una herencia para React.Component y le da a su componente acceso a las funciones de React.Component.

El componente también requiere un método render(), este método devuelve HTML.

Ejemplo
Cree un componente de Clase llamado ComponentHeader

```
class ComponentHeader extends React.Component {
  render() {
    return <h2>Hola, Yo soy el Header!</h2>;
  }
}
```

Componente de función

Un componente de función también devuelve HTML y se comporta de la misma manera que un componente de clase, pero los componentes de función se pueden escribir usando mucho menos código, son más fáciles de entender.

Ejemplo

Cree un componente de función llamado ComponentHeader

```
class ComponentHeader() {
  render() {
    return <h2>Hola, Yo soy el Header!</h2>;
  }
}
```

Renderizar un componente

Ahora su aplicación React tiene un componente llamado ComponentHeader, que devuelve un <h2>elemento.

Para llamar o mostrar un componente: utilice *<NombreComponente />*

Ejemplo

Muestre el ComponenteHeader en un elemento con Id "root":

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

Props en React

Los *props*, son argumentos (datos), que se envían al componente.

- Los ***props*** son definidos como uno o más atributos en un componente.
- Los ***props*** es un ***objeto*** que se define como argumento en la función.

Ejemplo

Use un atributo para pasar un color al componente ComponentHeader y utilícelo en la función render():

```
function ComponentHeader(props) {
  return <h2>This es header {props.color}!</h2>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<ComponentHeader color="red"/>);
```

Ejemplo

Cree una variable nombrada nombreLogo y envíela al componente ComponentLogo:

```
function ComponentLogo(props) {
    return <h2>This es Logo { props.logo }!</h2>;
  }

  function ComponentHeader() {
    const nombreLogo = "logo1.jpg";
    return (
      <>
        <h1>This is ComponentHeader</h1>
        <ComponentLogo logo={ nombreLogo } />
      </>
    );
  }

  const root = ReactDOM.createRoot(document.getElementById('root'));
  root.render(<ComponentHeader />);
```

Ejemplo

Cree un objeto llamado logoInfoy envíelo al ComponenteLogo:

```
function ComponentLogo(props) {
  return <h2>This is Logo: { props.logo.name }!</h2>;
}

function ComponentHeader() {
  const logoInfo = { name: "logo1.jpg", size: "50kb" };
  return (
    <>
      <h1>This is header</h1>
      <ComponentLogo logo={ logoInfo } />
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<ComponentHeader />);
```

Children en React

Los *children*, son argumentos (datos), que se envían al componente como contenido.

- Los ***children*** son definidos como uno o más elementos de un componente.
- Los ***children*** es un ***objeto*** que se define como argumento en la función.

Ejemplo

```
<IconButton> Texto Boton </IconButton>
```

```
function IconButton({ children }) {
  return (
      <button>
          <i class="target-icon" />
          {children}
      </button>
  );
}
```

Componentes dentro de Componentes

Podemos referirnos a componentes dentro de otros componentes:

Ejemplo

Utilice el componente ComponentLogo dentro del componente ComponentHeader:

```
function ComponentLogo() {
  return <h2>This is Logo!</h2>;
}

function ComponentHeader() {
  return (
    <>
      <h1>This es Header</h1>
      <ComponentLogo />
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<ComponentHeader />);
```

Componentes en archivos

React tiene que ver con la reutilización del código, y se recomienda dividir sus componentes en archivos separados.

Para hacer eso, cree un nuevo archivo con una .js extensión de archivo y coloque el código dentro:

Ejemplo

Este es el nuevo archivo, lo llamamos "ComponentLogo.jsx":

```
function ComponentLogo() {
  return <h2>This is Logo!</h2>;
}

export default ComponentLogo;
```

Ejemplo

Ahora importamos el archivo " ComponentLogo.jsx" en la aplicación, y podemos usar el ComponentLogo componente como si se hubiera creado aquí.

```
import ComponentLogo from './ComponetLogo.js';

function ComponentHeader() {
  return (
    <>
      <h1>This es Header</h1>
      <ComponentLogo />
    </>
  );
}
```

Evaluación de competencia

1. Desarrolle una aplicacion en React que tenga 3 divs en el archivo index.html   y crea 3 componentes que devuelvan un mensaje “Desde el componente X” y que se renderice cada componente en cada div.
2. Modifique el ejemplo anterior utilizando componentes de clase.
3. Desarrolle una aplicación en React que contenga 2 componentes anidados, crea los atributos nombre y dirección en el componente 1 y el componente 2 recibe y visualiza el nombre y dirección en dos h1.
4. Desarrolle una aplicación en React que contenga 5 componentes anidados, es decir el componente1 contiene al componente 2 este al componente 3 y este al componente 4 y este contiene al componente 5. Y se desea pasar un objeto con tres propiedades desde el componente 1 al componente 5 quién visualizará los tres datos.
5. Desarrolle una aplicación en React con 2 componentes un componente padre y un hijo, y pasar datos del componente hijo al padre, y que el dato pueda ser renderizado por el componente padre.
6. Investigue si se tiene 3 componentes un componente padres y dos hijos, como podría pasar datos del componente hermano 1 al hermano2.