

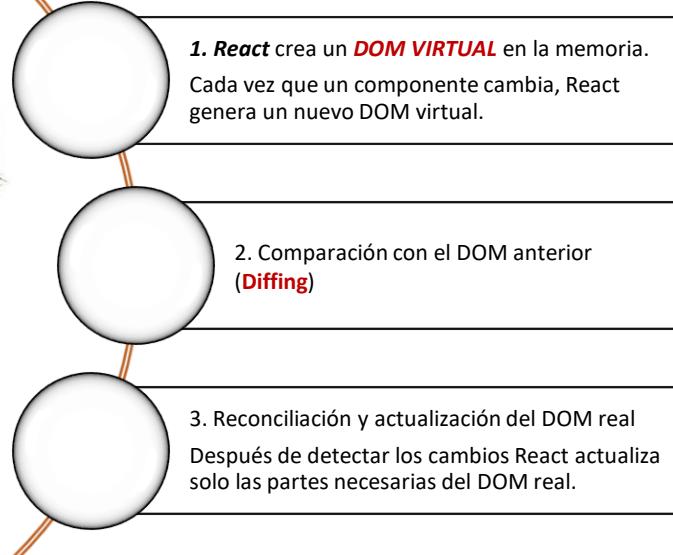
Actividad de Aprendizaje N° 07A

Componentes, JSX, TypeScript y Estilos en React

1. Renderizar HTML en React



¿Cómo Renderiza el DOM 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)
```

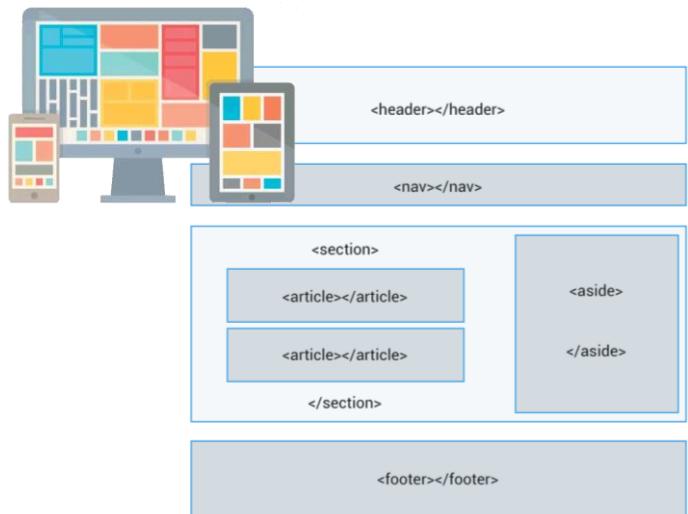
La función ReactDOM.render() tiene dos argumentos:

1. Component HTML
2. Element HTML a ser modificado.

2. Componentes en React

¿Qué es un Componente?

Un **componente** es una: **unidad de interfaz o unidad lógica REUTILIZABLE** bien definidas que permiten *ofrecer y/o solicitar funcionalidades o servicios*.



Web Components



- Secciones de la pantalla de interfaz de usuario.
- En las **páginas web**, una buena práctica es dividirlo en **componentes**
- Secciones del **html semántico**

Creación de Componentes en React

Un componente se crea en un archivo .js, .jsx, .ts, .tsx

El nombre de un componente en React debe **empezar con una letra Mayúscula**.

Sintaxis - Creación

```
function App() {
  //Código JSX
  return (
    <>
      <h1>Soy el componente App</h1>
      {/* Código html y css */}
    </>
  )
}
export default App
```

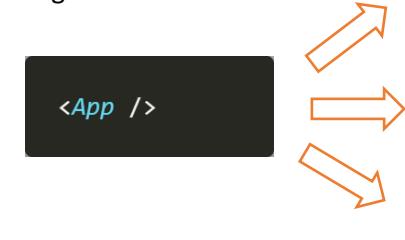
Sintaxis - Invocación:

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

Estructura de un Componente

Todo componente se subdivide en 3 partes:

1. Código html
2. Código JavaScript
3. Código CSS



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

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

export default App
```

3. Desarrollo de Componentes

Son de dos tipos:

- Componentes de Clase
- Componentes de Función

Componente de clase

Cree un componente de Clase llamado ComponentHeader

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

Componente de función

Los componentes de función se pueden escribir usando mucho menos código, son más fáciles de entender.

Ejemplo

Cree un componente de función llamado ComponentHeader

```
const ComponentHeader = function(){
    return (
        <h2>ComponentHeader</h2>
    )
}
```

Creación del Componente

1. Crear un archivo en *src/components/Component01.jsx*
2. En *Component01.jsx* digitar el código:

```
const Componente1 = () => {
    return (
        <div>
            <h4>Componente1</h4>
        </div>
    )
}
```

4. JSX - JavaScript eXtensible

¿Qué es JSX?

JSX es un lenguaje que mezcla de *Javascript* y *HTML* fue creada por *Facebook* para el uso con su *librería React*. JSX requiere transformarse a *JavaScript* mediante un transpilador como *Babel*.

Nota. En vez de JSX podemos usar Typescript

Variable o Constante

Las variables son como en JavaScript, pero permite almacenar código html.

Variable *como en JS*

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

Variable *contiene etiquetas html*:

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

Nota. Sin comillas

Expresiones en JSX

En JSX puedes escribir expresiones dentro de **llaves {}** que se ejecuta como Javascript

En el operador {} se puede:

- Insertar las variables
- Insertar atributos en etiquetas
- 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 element = <h1>Año!! {2000 + 24} </h1>
```

Se agregó: {2000 + 24} que devuelve la suma //2024.

Insertar un bloque grande de HTML

Usar paréntesis:

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

Envolver con una etiqueta de nivel superior

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

Elemento div como superior

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

Un fragment

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

Los elementos deben estar cerrados

Los elementos HTML **deben tener su etiqueta de apertura y cierre**. Algunas etiquetas no tienen etiqueta de cierre, **utilice “/”**.

Ejemplo

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

Condicional IF

React permite la sentencia if, pero no dentro del HTML CSS. Podemos renderizar condicionalmente componentes:

Ejemplo:

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

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

Operador Ternario

condicion ? expresionTrue : expresionFalse

```
const x = 5;

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

Operador &&

Permite devolver un solo valor se usa en vez del operador ternario.

condicion && expresionTrue

```
const resultado=error &&
(<div>
    <h3>Formato de email incorrecto</h3>
</div>)
```

Bucles Map

Permite recorrer un array u un objeto.

```
{students.map(
    (student)=><ItemEstudiante id={student.id}
name={student.name} city={student.city} />
)}
```

Observaciones de Codificación en JSX

- JSX nos permite renderizar el DOM sin: *createElement()* o *.appendChild()*.
- JSX convierte etiquetas HTML en elementos reactivos.

5. TypeScript

¿Qué es TypeScript?

TypeScript es un lenguaje JavaScript con tipado estático opcional desarrollado por Microsoft. Permite detectar errores antes de la ejecución y mejorar la estructura del código. requiere transformarse a JavaScript mediante un transpilador como Babel.

Variable o Constante

Las variables son como en JavaScript, pero permite almacenar código html.

```
const usuario:string = 'Jaime'
const myId:string = 'titulo'
```

```
const element: JSX.Element = <h1 id={myId}>Bienvenido {usuario}!!!</h1>;
```

Traducción de Componente a TypeScript

Si se tiene un componente JSX

```
const PrimerComponente = ({ number, setNumber }) => {

  return (
    <>
      <div>{number}</div>

      <button
        onClick={() => setNumber(prev => prev+1)}
      >
        ADD
      </button>
    </>
  )
}

export default PrimerComponente
```

Componente en TypeScript

```
import React, { Dispatch, SetStateAction } from 'react'

interface PrimerProps {
  number: number
  setNumber: Dispatch<SetStateAction<number>>
}

const PrimerComponente:React.FC<PrimerProps> = ({ number, setNumber }) => {

  return (
    <>
      <div>{number}</div>

      <button
        onClick={() => setNumber(prev => prev+1)}
      >
        ADD
      </button>
    </>
  )
}

export default PrimerComponente
```

6. Diseñar Componentes en React

Hay muchas formas de diseñar un componente en React con CSS o una librería CSS. Con CSS hay tres formas de hacerlo:

- hojas de estilo CSS
- Módulos CSS
- estilo en línea

Hojas de Estilos CSS:

Crear archivo con extensión *.css.

./App.css

```
.myclass{  
    color:blue;  
}
```

Importar en el archivo CSS y utilizar el estilo con **ClassName**

```
import './App.css'  
function App() {  
    return (  
        <>  
            <h1 className="myclass">Soy el componente App</h1>  
        </>  
    )  
}  
export default App
```

Estilos en Módulos:

Crear archivo con extensión *.module.css..

../css/estilo.module.css

```
.clase1{  
    background-color: blueviolet;  
    color: aquamarine;  
    padding: 50px;  
}
```

Importar y llamar como objeto

```
import estilos from '../css/estilo.module.css'  
const Blogs = function () {  
    return <h1 className={estilos.clase1}>Blogs me aquí</h1>  
}  
  
export default Blogs;
```

Estilos Inline:

Utilizar estilo como objeto {{ }}

```
const NoPage = function() {  
    return <h1 style={{color: "red"}}>404 Página NO encontrada</h1>  
}  
  
export default NoPage;
```

Utilizar varios estilos con un nombre de objeto

```
const Contact = function() {
  const myStyle={
    color: "orange",
    backgroundColor: "black",
    padding: "30px",
    fontFamily: "forte"
  };
  return <h1 style={myStyle}>Contact me aqui</h1>
}

export default Contact;
```

7. Librería tailWind CSS en React

1. Instalación

Ejecute el comando.

```
>npm install -D tailwindcss postcss autoprefixer
>npx tailwindcss init -p
```

Nota: debe estar en la carpeta del proyecto Vite. El comando npx init genera archivos *tailwind.config.js* como *postcss.config.js*.

2. Configuración

Configurar los path de plantilla

Agregar los paths de todos los archivos que utilizarán tailWind en *tailwind.config.js*.

```
module.exports = {
  content: [
    './index.html',
    './src/**/*.{js,jsx}'
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

3. Configurar las directivas Tailwind a su CSS

Agregue las directivas @tailwind en *./src/index.css*.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

8. Props y Children de Componentes

Permiten transferir datos entre componentes.

Props

Los *props*, es un objeto argumento (datos), que se envían a la función del componente.

- Los *props* son definidos como uno o más atributos en un componente.

Ejemplo

Crear un componente padre y un componente hijo desde el padre enviar un dato hacia el componente hijo.

```
const Padre = () => {
  let nombre = "Jaime";
  return (
    <>
      <div>Soy el Componente Padre</div>
      <Hijo dato={nombre} />
    </>
  )
}

export default Padre
```

```
const Hijo = (props) => {
  return (
    <>
      <div>Soy el componente Hijo</div>
      <p>{props.dato}</p>
    </>
  )
}

export default Hijo
```

Children

Los **children**, es un argumento (dato), que se envían al componente hijo como contenido.

Ejemplo

```
const Padre = () => {
  let nombre = "Jaime";
  return (
    <>
      <div>Soy el Componente Padre</div>
      <Hijo>Información Children</Hijo>
    </>
  )
}

export default Padre
```

```
const Hijo = ({children}) => {
  return (
    <>
      <div>Soy el componente Hijo</div>
      <p>{children}</p>
    </>
  )
}

export default Hijo
```

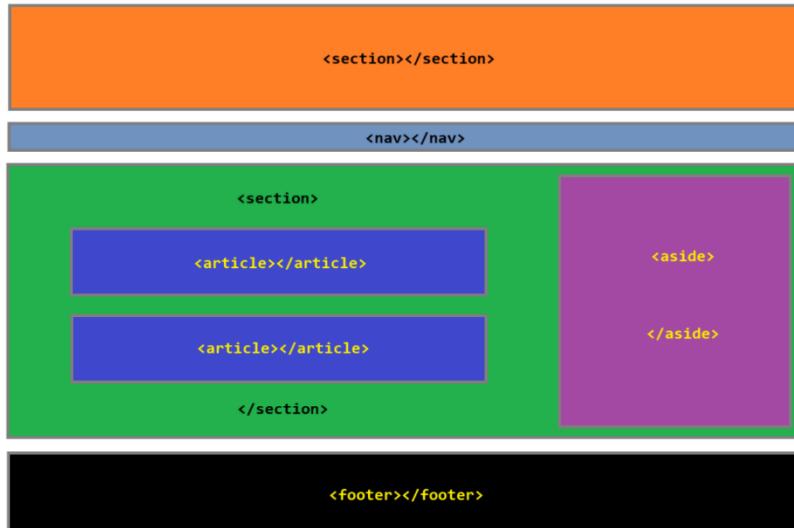
Componentes dentro de Componentes

Podemos referirnos a componentes dentro de otros componentes:

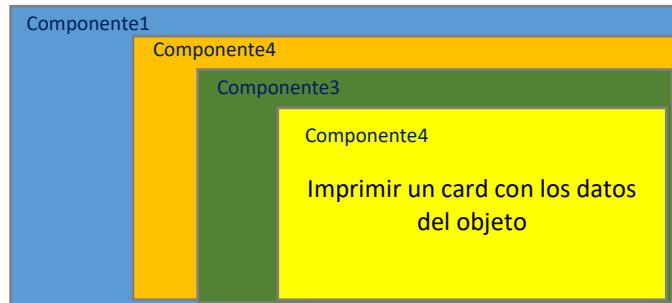
```
<App />  
<Padre />  
<Hijo />
```

Ejercicios de Laboratorio

1. Desarrolle una aplicación web en React con un diseño responsive diferente para pc, Tablet y móvil considerando la creación de 7 componentes (uno por sección) con diseño basado en css puro. El diseño inicial para pc es la siguiente.



2. Desarrolle una aplicación en React que contenga 4 componentes anidados, es decir el componente1 contiene al componente 2 hasta el componente 4. Y se desea pasar un objeto `{nombre="Jaime", dirección="Jr. Junin 450", ciudad="Huancayo"}` desde el componente 1 al componente 4. Los datos recibidos se visualizarán en un card.



3. 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.
4. Desarrolle 3 componentes un componente padre y dos hijos, pasar datos un nombre y apellido del componente hermano 1 al hermano2.
5. Desarrollar componentes en React para renderizar los datos de un objeto de 4 estudiantes id, name y city. Los datos deberán ser presentados en una tabla con estilos CSS.

Lista de estudiantes

Id	Name	City	Action
1	Jose Lazo	Huancayo	<input type="button" value="Ver"/>
2	Ana Trelles	Lima	<input type="button" value="Ver"/>
3	Pedro Gonzales	Arequipa	<input type="button" value="Ver"/>
4	Rosa Soto	Trujillo	<input type="button" value="Ver"/>