



JavaScript y React

Clase 11

KOICA

IGU HANDONG GLOBAL
UNIVERSITY



UNA

OBJETIVOS DE LA CLASE 11

— — —

- Manejar los conceptos de Controles FORM, BUTTON y TEXT.
- Manejar el concepto de Control PASSWORD.
- Manejar el concepto de Eventos: load y DOMContentLoaded.
- Manejar el concepto de Parámetro del método asociado al addEventListener.
- Manejar el formato JSX.
- Implementar los códigos de ejemplos propuestos en clase.



Controles FORM, BUTTON y TEXT

- Hasta ahora hemos visto cómo crear un formulario con controles de tipo BUTTON. Agregamos un control de tipo TEXT (permite al operador cargar caracteres por teclado).
- Veremos la importancia de definir un id a todo control de un formulario.
- Con un ejemplo veremos estos controles.

Ejemplo

Confeccionar un formulario que permita ingresar el nombre y edad de una persona

```
<body>

    <form>
        <p>Ingrese su nombre: <input type="text"
id="nombre"></p>
        <p>Ingrese su edad: <input type="text"
id="edad"></p>
        <p><input type="button" value="Confirmar"
onClick="mostrar()"></p>
    </form>

    <script>
        function mostrar() {
            let nom =
document.getElementById( 'nombre' ).value;
            let ed = document.getElementById( 'edad' ).value;
            alert( 'Ingresó el nombre:' + nom );
            alert( 'Y la edad:' + ed );
        }
    </script>

</body>
```

Controles FORM, BUTTON y TEXT

— — —

- En este problema tenemos cuatro controles: 1 FORM, 1 BUTTON, 2 TEXT. El evento que se dispara al presionar el botón se llama mostrar.
- La función 'mostrar' accede a los contenidos de los dos controles de tipo TEXT:

```
let nom = document.getElementById('nombre').value;  
let ed = document.getElementById('edad').value;
```

Controles FORM, BUTTON y TEXT

— — —

```
let nom = document.getElementById('nombre').value;  
let ed = document.getElementById('edad').value;
```

- Para hacer más clara la función guardamos en dos variables auxiliares los contenidos de los controles de tipo TEXT.
- La propiedad "id" es un identificador único para cualquier elemento HTML que luego nos permite desde Javascript acceder a dicho elemento.
- El método getElementById nos retorna una referencia del objeto HTML que le pasamos como parámetro, a partir de este objeto accedemos a la propiedad value que almacena el valor ingresado por el operador en el control TEXT.

Controles FORM, BUTTON y TEXT

— — —

- Luego de extraer los valores ingresados por el operador los mostramos utilizando la función alert:

```
let nom = document.getElementById('nombre').value;  
let ed = document.getElementById('edad').value;  
alert('Ingresó el nombre:' + nom);  
alert('Y la edad:' + ed);
```

Problema

Crear un programa que permita cargar un entero en un text y al presionar un botón nos muestre dicho valor elevado al cubo (emplear la función alert).

```
<body>

    <form>
        <p>Ingrese un valor:<input type="text"
id="num"></p>
        <p><input type="button" value="Calcular
cubo" onClick="calcularCubo()"></p>
    </form>

    <script>
        function calcularCubo() {
            let v =
document.getElementById('num').value;
            v = parseInt(v);
            let cubo = v * v * v;
            alert(cubo);
        }
    </script>

</body>
```


Problema

— — —

Cargar dos números en objetos de tipo text y al presionar un botón, mostrar el mayor.

```
<body>

  <form>
    <p>Ingrese primer valor:
      <input type="text" id="num1"></p>
    <p>Ingrese segundo valor:
      <input type="text" id="num2"></p>
    <p><input type="button" value="mostrar mayor"
onClick="mostrarMayor()"></p>
  </form>

  <script>
    function mostrarMayor() {
      let num1 =
parseInt(document.getElementById('num1').value);
      let num2 =
parseInt(document.getElementById('num2').value);
      if (num1 > num2) {
        alert('El mayor es ' + num1);
      } else {
        alert('El mayor es ' + num2);
      }
    }
  </script>

</body>
```

Problema

— — —

Cargar un nombre y un apellido en dos campos text. Al presionar un botón, concatenarlos y mostrarlos en un tercer text (Tener en cuenta que podemos modificar la propiedad value de un objeto TEXT cuando ocurre un evento)

```
<body>

    <form>
        <p>Ingrese nombre:<input type="text"
id="nombre"></p>
        <p>Ingrese apellido:<input type="text"
id="apellido"></p>
        <p><input type="button" value="Concatenar
datos ingresados" onClick="concatenar()"></p>
        <p><input type="text" id="resultado"></p>
    </form>

    <script>
        function concatenar() {
            let nom =
document.getElementById('nombre').value;
            let ape =
document.getElementById('apellido').value;

document.getElementById('resultado').value = nom +
ape;

        }
    </script>

</body>
```

Control PASSWORD

- Este control HTML es una variante de la de tipo "TEXT". La diferencia fundamental es que cuando se carga un texto en el campo de edición sólo muestra asteriscos en pantalla, es decir, es fundamental para el ingreso de claves y para que otros usuarios no vean los caracteres que tipeamos.
- La mayoría de las veces este dato se procesa en el servidor, pero en el cliente (es decir, en el navegador) podemos verificar si ha ingresado una cantidad correcta de caracteres, por ejemplo.

Ejemplo

Codificar una página que permita ingresar un password y luego muestre un ventana de alerta si tiene menos de 5 caracteres.

```
<body>

    <form>
        Ingrese una clave:
        <input type="password" id="clave">
        <br>
        <input type="button" value="Confirmar"
onClick="verificar()">
    </form>

    <script>
        function verificar() {
            let clave =
document.getElementById('clave').value;
            if (clave.length < 5) {
                alert('La clave no puede tener menos
de 5 caracteres!!!');
            } else {
                alert('Largo de clave correcta');
            }
        }
    </script>

</body>
```

Control PASSWORD

- En este problema debemos observar que cuando ingresamos caracteres dentro del campo de edición sólo vemos asteriscos, pero realmente en memoria se almacenan los caracteres tipeados. Si queremos mostrar los caracteres ingresados debemos acceder mediante el método `getElementById` a la marca HTML clave:

```
let clave = document.getElementById('clave').value;
```

Control PASSWORD

— — —

- Normalmente, a este valor no lo mostraremos dentro de la página, si no se perdería el objetivo de este control (ocultar los caracteres tipeados).
- Si necesitamos saber la cantidad de caracteres que tiene un string accedemos a la propiedad `length` que retorna la cantidad de caracteres.

```
if (clave.length<5)
```

Problema

Disponer dos campos de texto tipo password. Cuando se presione un botón mostrar si las dos claves ingresadas son iguales o no (es muy común solicitar al operador el ingreso de dos veces de su clave para validar si las escribió correctamente, esto se hace cuando se crea un password para el ingreso a un sitio o para el cambio de una existente).

Tener en cuenta que podemos emplear el operador `==` para ver si dos string son iguales.

```
<body>

  <form>
    Ingrese clave:
    <input type="password" id="clave1">
    <br> Repita la clave:
    <input type="password" id="clave2">
    <br>
    <input type="button" value="Verificar clave"
onClick="verificar()">
  </form>

  <script>
    function verificar() {
      let clave1 =
document.getElementById('clave1').value;
      let clave2 =
document.getElementById('clave2').value;
      if (clave1 == clave2) {
        alert('Las dos claves ingresadas son
iguales');
      } else {
        alert('Las dos claves ingresadas son
distintas');
      }
    }
  </script>

</body>
```

Eventos: load y DOMContentLoaded

- Como hemos visto hasta ahora uno de los usos principales de JavaScript es la implementación de algoritmos que reaccionan a eventos que se producen en una página web.
- Cuando se termina de cargar completamente una página web se dispara el evento onload, cuando se presiona un botón de tipo submit se dispara el evento onsubmit, cuando movemos la flecha del mouse se dispara onmousemove y otra gran cantidad de eventos que nos informa el navegador y nosotros podemos capturarlos y hacer un algoritmo a nuestra medida.

Eventos: load y DOMContentLoaded

- A lo largo de estos más de 25 años de la existencia de JavaScript, se han implementado tres formas distintas de capturar los eventos que emite el navegador.

Eventos definidos directamente en la marca HTML

- Esta metodología está en desuso, de todos modos muchos sitios antiguos implementan esta técnica.

Problema

Implementar un formulario que solicite la carga del nombre de usuario y su clave. Mostrar un mensaje si no se ingresan datos en alguno de los dos controles.

```
<body>

    <form method="post" action="procesar.php" onsubmit="validar();"
    id="formulario1">

        Ingrese nombre:
        <input type="text" id="usuario" name="usuario" size="20">
        <br> Ingrese clave:
        <input type="password" id="clave" name="clave" size="20">
        <br>

        <input type="submit" id="confirmar" name="confirmar"
        value="Confirmar">
    </form>

    <script>
        function validar() {
            let usu = document.getElementById("usuario").value;
            let cla = document.getElementById("clave").value;
            if (usu.length == 0 || cla.length == 0) {
                alert('El nombre de usuario o clave está vacío');
                return false;
            } else
                return true;
        }
    </script>

</body>
```

Eventos definidos directamente en la marca HTML

- Lo primero que tenemos que ver que la marca form define la propiedad onsubmit y le asigna el nombre de la función JavaScript que debe llamarse previo a que el navegador empaquete todos los datos del formulario y los envía al servidor para ser procesados:

```
<form method="post" action="procesar.php" onsubmit="validar();">
```

- Como vemos debemos indicar el nombre de la función y los paréntesis (en este caso no se envían parámetros por lo que los paréntesis van abiertos y cerrados).

Eventos definidos directamente en la marca HTML

- En el bloque JavaScript debemos implementar la función validar donde extraemos los valores de cada control y verificamos si alguno de los dos no tiene cargado caracteres, en caso que suceda esto mostramos un mensaje y retornamos un false para que el navegador no envíe los datos del formulario al servidor. Si la función retorna true los datos del formulario son enviados por el navegador al servidor:

Eventos definidos directamente en la marca HTML

```
function validar() {  
    let usu = document.getElementById("usuario").value;  
    let cla = document.getElementById("clave").value;  
    if (usu.length == 0 || cla.length == 0) {  
        alert('El nombre de usuario o clave está vacío');  
        return false;  
    } else  
        return true;  
}
```

- Esta metodología de inicializar eventos debe ser evitada en todo lo posible.

Eventos definidos accediendo a propiedades del objeto

- Esta metodología es todavía ampliamente utilizada ya que la mayoría de los navegadores lo implementan en forma similar.

Problema

Implementar un formulario que solicite la carga del nombre de usuario y su clave. Mostrar un mensaje si no se ingresan datos en alguno de los dos controles.

```
<body>

    <form method="post"
action="procesar.php"
id="formulario1">

        Ingrese nombre:
        <input type="text"
id="usuario" name="usuario"
size="20">

        <br> Ingrese clave:
        <input
type="password" id="clave"
name="clave" size="20">

        <br>
        <input type="submit"
id="confirmar"
name="confirmar"
value="Confirmar">

    </form>
```

```
<script>

    window.onload = inicio;

    function inicio() {

document.getElementById("formulario1").onsubmit
= validar;

    }

    function validar() {

        let usu =
document.getElementById("usuario").value;

        let cla =
document.getElementById("clave").value;

        if (usu.length == 0 || cla.length ==
0) {

            alert('El nombre de usuario o
clave está vacío');

            return false;

        } else

            return true;

    }

</script>

</body>
```


Eventos definidos accediendo a propiedades del objeto

- Con esta segunda metodología vemos que el código HTML queda limpio de llamadas a funciones JavaScript y todo el código queda dentro del bloque del script (pudiendo luego llevar todo este bloque a un archivo externo *.js)
- Lo primero que vemos es inicializar la propiedad onload del objeto window con el nombre de la función que se ejecutará cuando finalice la carga completa de la página, es importante notar que a la propiedad onload le asignamos el nombre de la función y NO debemos disponer los paréntesis abiertos y cerrados (ya que no se está llamando a la función sino le estamos pasando la dirección o referencia de la misma).

Eventos definidos accediendo a propiedades del objeto

— — —

```
window.onload = inicio;
```

- La función inicio es llamada por el objeto window cuando se termina de cargar la página. En esta función obtenemos la referencia del objeto formulario1 mediante el método getElementById e inicializamos la propiedad onsubmit con el nombre de la función que será llamada cuando se presione el botón submit del formulario:

```
function inicio() {  
    document.getElementById("formulario1").onsubmit = validar;  
}
```

Eventos definidos accediendo a propiedades del objeto

- Por último tenemos la función validar que verifica si los dos controles del formulario están cargados:

```
function validar() {  
    let usu = document.getElementById("usuario").value;  
    let cla = document.getElementById("clave").value;  
    if (usu.length == 0 || cla.length == 0) {  
        alert('El nombre de usuario o clave está vacío');  
        return false;  
    } else  
        return true;  
}
```

Eventos definidos accediendo a propiedades del objeto

- La misma metodología pero utilizando funciones anónimas para cada evento el código ahora queda más conciso:

```
<script>
  window.onload = function() {
    document.getElementById("formulario1").onsubmit = function() {
      let usu = document.getElementById("usuario").value;
      let cla = document.getElementById("clave").value;
      if (usu.length == 0 || cla.length == 0) {
        alert('El nombre de usuario o clave está vacío');
        return false;
      } else
        return true;
    }
  }
</script>
```

Eventos definidos accediendo a propiedades del objeto

- Analicemos un poco el código implementado, a la propiedad onload del objeto window le asignamos una función anónima:

```
window.onload = function() {  
    ...  
}
```

- En la implementación de la función anónima inicializamos la propiedad onsubmit del objeto formulario1 con otra función anónima:

```
document.getElementById("formulario1").onsubmit = function() {  
    ...  
}
```

- Esta sintaxis de funciones anónimas es ampliamente utilizado.

Modelo de eventos definidos por W3C (World Wide Web Consortium)

— — —

- Este modelo de eventos se basa en la implementación de un método para todos los objetos que nos permite registrar eventos. La sintaxis del método es:

```
addEventListener(evento, método a ejecutar);
```

- Veamos cómo implementamos el problema anterior utilizando este nuevo modelo de eventos:

Problema

Implementar un formulario que solicite la carga del nombre de usuario y su clave. Mostrar un mensaje si no se ingresan datos en alguno de los dos controles.

```
<body>

    <form method="post"
action="procesar.php"
id="formulario1">

        Ingrese nombre:
        <input type="text"
id="usuario" name="usuario"
size="20">

        <br> Ingrese clave:
        <input
type="password" id="clave"
name="clave" size="20">
        <br>
        <input type="submit"
id="confirmar"
name="confirmar"
value="Confirmar">

    </form>
```

```
<script>

    window.addEventListener('load', inicio);
    function inicio() {
document.getElementById("formulario1").addEventL
istener('submit', validar);
    }
    function validar(evt) {
        let usu =
document.getElementById("usuario").value;
        let cla =
document.getElementById("clave").value;
        if (usu.length == 0 || cla.length ==
0) {

            alert('El nombre de usuario o
clave está vacío');

            evt.preventDefault();

        }
    }
}
</script>
</body>
```

Modelo de eventos definidos por W3C (World Wide Web Consortium)

- Lo primero que vemos es que en vez de inicializar la propiedad onload procedemos a llamar al método addEventListener:

```
window.addEventListener('load', inicio);
```

- El primer parámetro es un string con el nombre del evento a inicializar, el segundo parámetro es el nombre de la función a ejecutar.

Modelo de eventos definidos por W3C (World Wide Web Consortium)

- Cuando se carga completamente la página el objeto window tiene la referencia al método que se debe llamar, en nuestro caso se llama inicio. La función inicio obtiene la referencia del objeto formulario1 y procede a registrar el evento submit indicando en el segundo parámetro el nombre de la función que debe ejecutarse:

```
function inicio() {  
    document.getElementById("formulario1").addEventListener('submit', validar);  
}
```

- El código de la función validar se modifica, llega como parámetro una referencia al evento y mediante este llamamos al método preventDefault si queremos que no se envíen los datos al servidor:

Modelo de eventos definidos por W3C (World Wide Web Consortium)

— — —

```
function validar(evt) {  
    let usu = document.getElementById("usuario").value;  
    let cla = document.getElementById("clave").value;  
    if (usu.length == 0 || cla.length == 0) {  
        alert('El nombre de usuario o clave está vacío');  
        evt.preventDefault();  
    }  
}
```

- Este modelo de eventos está siendo ampliamente implementado por los navegadores modernos. El problema es el IE8 o inferiores que no lo implementa de esta forma.

Evento DOMContentLoaded

- Evento 'load': El evento load se dispara cuando el contenido del archivo HTML y las referencias a todos los recursos asociados (imágenes, hojas de estilo etc.) se han cargado en memoria del navegador.
- Evento 'DOMContentLoaded': El evento DOMContentLoaded se dispara cuando el contenido del archivo HTML se ha cargado en el navegador, sin necesitar esperar imágenes, hojas de estilo etc.
- Es muy conveniente en la mayoría de las veces inicializar script de JavaScript utilizando éste evento en lugar del evento 'load'. Todos los navegadores modernos disponen del evento 'DOMContentLoaded'.
- En el problema anterior solo cambiamos la referencia del evento load por DOMContentLoaded:

Problema

Implementar un formulario que solicite la carga del nombre de usuario y su clave. Mostrar un mensaje si no se ingresan datos en alguno de los dos controles.

```
<body>

    <form method="post"
action="procesar.php"
id="formulario1">

        Ingrese nombre:
        <input type="text"
id="usuario" name="usuario"
size="20">

        <br> Ingrese clave:
        <input
type="password" id="clave"
name="clave" size="20">

        <br>
        <input type="submit"
id="confirmar"
name="confirmar"
value="Confirmar">

    </form>
```

```
<script>

    window.addEventListener('DOMContentLoaded',
inicio);

    function inicio() {

document.getElementById("formulario1").addEventL
istener('submit', validar);

    }

    function validar(evt) {

        let usu =
document.getElementById("usuario").value;

        let cla =
document.getElementById("clave").value;

        if (usu.length == 0 || cla.length ==
0) {

            alert('El nombre de usuario o
clave está vacío');

            evt.preventDefault();

        }

    }

</script>

</body>
```

Parámetro del método asociado al addEventListener

- Hasta ahora siempre hemos asociado una función distinta para procesar los eventos de distintos elementos HTML. Pero veremos que está permitido asociar una única función a varios eventos de distintos objetos.
- Debemos definir un parámetro en la función a implementar que llega la referencia del objeto que emitió el evento.
- Implementaremos un panel con un conjunto de botones que nos permitan ingresar un valor numérico presionando botones.
- Asociaremos el click de cada botón con una única función.

Parámetro del método asociado al addEventListener

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de JavaScript</title>
  <meta charset="UTF-8">
  <style>
    .boton {
      width: 50px;
      height: 50px;
    }
    #resultado {
      font-size: 40px;
    }
  </style>
</head>
```

Parámetro del método asociado al addEventListener

— — —

```
<body>
```

```
  <input type="button" id="boton0" name="boton0" value="0" class="boton">
```

```
  <input type="button" id="boton1" name="boton1" value="1" class="boton">
```

```
  <input type="button" id="boton2" name="boton2" value="2" class="boton">
```

```
  <input type="button" id="boton3" name="boton3" value="3" class="boton">
```

```
  <input type="button" id="boton4" name="boton4" value="4" class="boton">
```

```
  <input type="button" id="boton5" name="boton5" value="5" class="boton">
```

```
  <input type="button" id="boton6" name="boton6" value="6" class="boton">
```

```
  <input type="button" id="boton7" name="boton7" value="7" class="boton">
```

```
  <input type="button" id="boton8" name="boton8" value="8" class="boton">
```

```
  <input type="button" id="boton9" name="boton9" value="9" class="boton">
```

```
  <br>
```

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

Parámetro del método asociado al addEventListener

— — —

```
<script>
    for (let x = 0; x <= 9; x++) {
        document.getElementById('boton' + x).addEventListener('click', presion);
    }

    function presion(evt) {
        document.getElementById('resultado').innerHTML =
            document.getElementById('resultado').innerHTML + evt.target.value;
    }
</script>

</body>

</html>
```


Parámetro del método asociado al addEventListener

- Asociamos los diez botones con la función presion y para reducir el código disponemos un for que se repita 10 veces y llamamos en la misma al método addEventListener (debemos ir obteniendo el id del botón concatenando 'boton' y el contador del for):

```
for (let x = 0; x <= 9; x++) {  
    document.getElementById('boton' + x).addEventListener('click', presion);  
}
```

Parámetro del método asociado al addEventListener

- Lo nuevo aparece en la función presion:

```
function presion(evt) {  
    document.getElementById('resultado').innerHTML =  
        document.getElementById('resultado').innerHTML + evt.target.value;  
}
```

Parámetro del método asociado al addEventListener

- La función presion tiene un parámetro llamado evt (podemos darle cualquier nombre) y el mismo es un objeto que tiene las siguientes propiedades:

target: Referencia del objeto que generó el evento (en nuestro ejemplo alguno de los 10 botones)

type: El nombre del evento (en nuestro caso click)

button: El botón del mouse presionado (0 = izquierdo, 1 = medio, 2 = derecho)

keyCode: El caracter del teclado presionado (en caso que corresponda)

shiftKey: true o false en caso de estar presionada esta tecla.

Parámetro del método asociado al addEventListener

- Como en este problema debemos ir concatenando el número presionado procedemos a obtener la referencia del div y asignarle el valor actual más la propiedad value del botón presionado:

```
document.getElementById('resultado').innerHTML =  
    document.getElementById('resultado').innerHTML + evt.target.value;
```

Crear tu segundo proyecto React con Vite

Abrí la terminal o consola de tu sistema operativo y escribí:

```
npm create vite@latest
```

Te pedirá:

- Nombre del proyecto: proyecto002
- Framework: seleccioná React
- Variant: elegí JavaScript (más adelante podés probar TypeScript)

Realizar el comando

```
cd proyecto002
```

 ¿Qué hace?

- cd significa “cambiar de carpeta” (viene del inglés change directory).
- Te metés dentro de la carpeta de tu nuevo proyecto.

 Es como decirle a la compu:

“Ahora quiero trabajar dentro de la carpeta proyecto002”.

 Después de esto, estás “dentro” del proyecto.

Realizar el comando

```
npm install
```

 ¿Qué hace?

Le dice a Node.js:

“Instalá todos los archivos que este proyecto necesita para funcionar”.

Realizar el comando

```
npm run dev
```

 ¿Qué hace?

Le dice a Vite:

“Arrancá el servidor para que yo pueda ver mi proyecto en el navegador”.

 Vite va a mostrar la página en `http://localhost:5173` (puede variar).

Modificar tu segunda app

— — —

Abrí src/App.jsx y reemplazá el código por:

```
function App() {  
  return (  
    <div>  
      <h1>¡Hola Mundo desde React con Vite!</h1>  
      <p>Mi primera aplicación de React</p>  
    </div>  
  );  
}  
  
export default App;
```



Guardá y mirá el navegador: los cambios se actualizan solos sin recargar.

Modificar tu segunda app

- La función App no retorna ni HTML, ni Javascript puro, es un nuevo formato propuesto por los creadores de React que luego de ser compilado se genera Javascript puro que lo pueden entender los navegadores.
- Hay ciertas reglas que debe cumplir el formato JSX (JavaScript XML), presentaremos algunas de ellas en este segundo ejercicio.
- Modifiquemos valor devuelto por la función App:

Modificar tu segunda app

— — —

```
import './App.css';

function retornarAleatorio() {
  return Math.trunc(Math.random() * 10);
}

function App() {
  const siglo = 21
  const persona = {
    nombre: 'Ivan',
    edad: 34
  }
}
```

```
return (
  <div>
    <h1>Título nivel 1</h1>
    <hr />
    <p>Estamos en el siglo {siglo}</p>
    <h3>Acceso a un objeto</h3>
    <p>{persona.nombre} tiene {persona.edad} años</p>
    <h3>Llamada a un método</h3>
    <p>Un valor aleatorio llamando a un método.</p>
    {retornarAleatorio()}
    <h3>Calculo inmediato de expresiones</h3>
    3 + 3 = {3 + 3}
  </div>
);
}

export default App;
```

Formato JSX

- La función App tiene por objetivo retornar el elemento JSX que representa la interfaz visual de la componente 'App' (por el momento desarrollaremos toda nuestra aplicación en una única componente, luego veremos que un programa se descompone en muchas componentes)
- Como vemos dentro del bloque de JSX podemos disponer etiquetas HTML tal como conocemos:

```
<h1>Título nivel 1</h1>
```

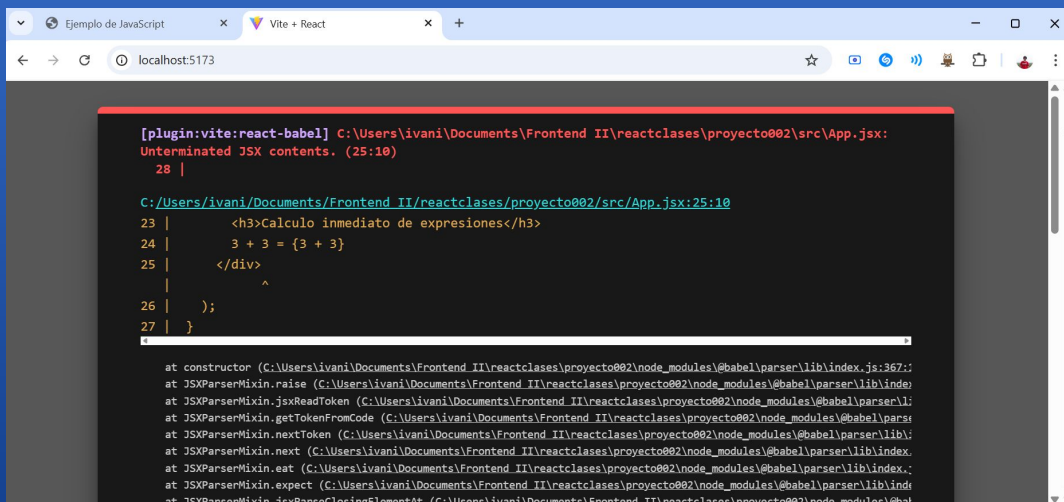
Formato JSX

- Una restricción de JSX es que siempre los elementos HTML deben tener su marca de comienzo y fin, y en el caso que solo tengan una etiqueta que es tanto de comienzo como fin debemos agregar el caracter '/':

```
<hr />
```

Formato JSX

- Si nos olvidamos de agregar la barra de cierre se genera un error cuando tratamos de compilar la aplicación:



```
[plugin:vite:react-babel] C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\src\App.jsx:
Unterminated JSX contents. (25:10)
 28 |
    |
    |
C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\src\App.jsx:25:10
 23 |         <h3>Calculo inmediato de expresiones</h3>
 24 |         3 + 3 = {3 + 3}
 25 |     </div>
    |         ^
 26 |     );
 27 | }

at constructor (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index.js:367:
at JSXParserMixin.raise (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index
at JSXParserMixin.jsxReadToken (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\li
at JSXParserMixin.getTokenFromCode (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\pars
at JSXParserMixin.nextToken (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\
at JSXParserMixin.next (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index
at JSXParserMixin.eat (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\index.
at JSXParserMixin.expect (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@babel\parser\lib\indi
at JSXParserMixin.jsxParseClosingElementAt (C:\Users\ivani\Documents\Frontend II\reactclases\proyecto002\node_modules\@ba
```

Formato JSX

- Dentro del bloque JSX podemos acceder a variables o constantes indicando entre llaves dicha variable o constante:

```
<p>Estamos en el siglo {siglo}</p>
```

Formato JSX

— — —

- Luego cuando se compila en lugar de la expresión se muestra el contenido de la variable o constante:



Formato JSX

- En forma similar podemos disponer expresiones para acceder a propiedades de un objeto previamente definido:

```
<p>{persona.nombre} tiene {persona.edad} años</p>
```

- Otra posibilidad en una expresión es hacer la llamada a otras funciones:

```
{retornarAleatorio()}
```

- Podemos inclusive disponer una operación que será ejecutada previo a su visualización:

```
3 + 3 = {3 + 3}
```

EJERCICIOS ADICIONALES PROPUESTOS

— — —





**¡MUCHAS
GRACIAS!**