JavaScript y React

Clase 13

KOICA

HANDONG GLOBAL UNIVERSITY

UNA

OBJETIVOS DE LA CLASE 13

- **Manejar Variables de estado de una componente mediante Hook (función useState)**
- **Manejar ES6, const, plantilla de cadena de caracteres, for of, map**
- **Implementar los códigos de ejemplos propuestos en clase.**

Crear tu cuarto proyecto React con Vite

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

```
npm create vite@latest
```

Te pedirá:

- Nombre del proyecto: proyecto004
- Framework: seleccioná React
- Variant: elegí JavaScript (más adelante podés probar TypeScript)

Realizar el comando

```
cd proyecto004
```

📌 ¿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 proyecto004”.

✅ 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).

Variables de estado de una componente mediante Hook (función useState)

- Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React.

- Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizarán en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora, etc.

Variables de estado de una componente mediante Hook (función useState)

- Debemos importar la función 'useState' si queremos administrar Hook de estados:

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

Problema

Crear un nuevo proyecto llamado: proyecto004.

Definir en la interfaz visual un botón que cada vez que se presione se actualice en pantalla un número aleatorio entre 0 y 9.

Modificar tu cuarta app

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

function App() {

  function generarAleatorio() {
    const v =
Math.trunc(Math.random() * 10);
    setNumero(v)
  }
```

```
const [numero, setNumero] = useState( 0);

  return (
    <div>
      <p>Número aleatorio: {numero}</p>
      <button
onClick={generarAleatorio}>Generar número
aleatorio</button>
    </div>
  );
}
export default App;
```

Variables de estado de una componente mediante Hook (función useState)

Llamamos a la función useState y definimos el valor inicial, en nuestro caso el valor entero cero, la función retorna un arreglo con dos valores que se almacenan en numero y setNumero (podemos definir cualquier nombre para estas dos variables):

```
const [numero, setNumero] =
useState(0);
```

Variables de estado de una componente mediante Hook (función useState)

La desestructuración de los dos elementos del arreglo que retorna useState es una característica del lenguaje Javascript (ES6)

En la posición cero del arreglo nos retorna la variable de estado y en la posición uno nos retorna una función que nos sirve para actualizar la variable de estado y las almacenamos en 'numero' y 'setNumero'.

¿Qué es la desestructuración en JavaScript?

Desestructuración es una forma rápida de sacar valores de un arreglo o de un objeto y guardarlos en variables.

```
const [a, b] = [10, 20];
```

Es lo mismo que hacer:

```
const a = 10;

const b = 20;
```

Variables de estado de una componente mediante Hook (función useState)

Cuando queremos cambiar el valor de la variable de estado llamamos a la función setNumero:

```
function generarAleatorio() {
    const v = Math.trunc(Math.random() * 10);
    setNumero(v)
  }
```

Variables de estado de una componente mediante Hook (función useState)

Y cuando queremos rescatar su valor accedemos directamente a la variable de estado 'numero':

```
<p>Número aleatorio: {numero}</p>
```

Luego cuando se presiona el botón, se dispara el evento generarAleatorio:

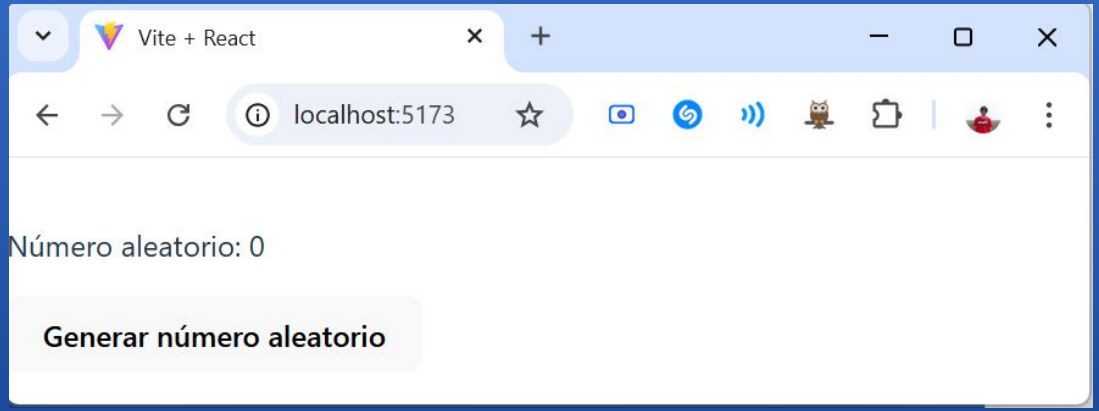
```
<button onClick={generarAleatorio}>Generar número aleatorio</button>
```

Variables de estado de una componente mediante Hook (función useState)

Cuando se llama a la función 'setNumero' modificamos la variable de estado, con esto la librería React se encarga de ejecutar nuevamente la graficación de la componente pero solo actualizando los estados cambiados y sin tener que redibujar la página completa (tener en cuenta que se ejectua nuevamente la función App, pero no se crea e inicializa nuevamente la variable de estado con cero. La documentación de React informa que se definió el nombre de la función 'useState' en lugar de 'createState' debido a que solo la primera vez que se llama a 'useState' se crea la variable de estado e inicializa en nuestro ejemplo con cero)

Variables de estado de una componente mediante Hook (función useState)

Tenemos como resultado en el navegador:



ES6

ES6 o también conocido como ECMAScript 2015 es una actualización del lenguaje de programación JavaScript que actualmente todos los navegadores ya implementan en 2021.

Todos los agregados a JavaScript definidos en este nuevo estándar ya se los está aplicando desde hace un tiempo con Node.js en el servidor, y en framework de cliente como React, Angular y Vue.

Estos agregados al lenguaje JavaScript nos facilitan la programación.

ES6

ES6 o también conocido como ECMAScript 2015 es una actualización del lenguaje de programación JavaScript que actualmente todos los navegadores ya implementan en 2021.

Todos los agregados a JavaScript definidos en este nuevo estándar ya se los está aplicando desde hace un tiempo con Node.js en el servidor, y en framework de cliente como React, Angular y Vue.

Estos agregados al lenguaje JavaScript nos facilitan la programación.

const

Hasta que llega ES6 la única forma de definir variables en JavaScript es mediante la palabra clave 'var':

```
var edad=12;

var sueldo=1000.50;

var titulo='Administración';
```

const

Con ES6 si sabemos que la variable nunca cambiará luego en lugar de definir con var debemos definir una constante con la palabra clave 'const':

```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>

<body>
    <script>
        const titulo = 'Administracion';
        const empleados = 10;
        const pi = 3.1416;
        alert(titulo);
        alert(empleados);
        alert(pi);
    </script>

</body>

</html>
```

ES6 - Plantillas de cadenas de caracteres

El manejo de cadenas de caracteres en JavaScript tradicional se emplean las comillas simples o dobles:

let titulo1='Administración';

let titulo2="Contabilidad";

Cuando se requieren generar otra cadena formada por varias variables de tipo cadena, numérica etc. normalmente se utiliza el operador + para concatenar.

Con ES6 se incorpora la posibilidad de definir cadenas de caracteres utilizando las comillas invertidas `.

ES6 - Plantillas de cadenas de caracteres

Emplearemos este tipo de cadenas cuando necesitemos sustituir el contenido de variables dentro de la cadena de caracteres sin tener que utilizar el operador de concatenación:

```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>

<body>

    <script>
        const valor1 =
parseInt(prompt('Ingrese primer valor:'));
        const valor2 =
parseInt(prompt('Ingrese segundo valor:'));
        document.write(`La suma de ${valor1}
y ${valor2} es ${valor1+valor2}`);
    </script>

</body>
</html>
```

ES6 - Plantillas de cadenas de caracteres

Para interpolar el contenido de una variable dentro de una plantilla de cadena de caracteres encerramos entre llaves la variable y le antecedemos el caracter $:

```
document.write(`La suma de ${valor1} y ${valor2} es ${valor1+valor2}`);
```

También es importante ver que dentro de las llaves podemos disponer una operación.

Podemos comprobar que es una sintaxis más clara a la de emplear el operador de concatenación:

```
document.write("La suma de "+valor1+" y "+valor2+" es "+(valor1+valor2));
```

ES6 - Estructura repetitiva 'for of'

Con ES6 disponemos una nueva estructura repetitiva para recorrer todos los elementos de un array, string, map, set etc.

En forma muy sencilla podemos acceder por ejemplo a cada caracter de un string:

```
const cadena='Hola Mundo';

for(let letra of cadena)

    document.write(`${letra}<br>`);
```

Como resultado de ejecutar éste algoritmo de JavaScript tenemos la impresión de cada caracter en una línea distinta en la página.

ES6 - Estructura repetitiva 'for of'

Podemos recorrer también un array:

```
const vector=[10, 40, 60, 5];
  let suma=0;
  for(let elemento of vector)
      suma+=elemento;
  document.write(`La suma de todos los elementos del vector es:${suma}`);
```

Problema

Se tiene el siguiente vector con 3 objetos literales con los datos de personas. Imprimir el nombre y edad de las personas mayores de edad:

```
const personas = [{
        nombre: 'juan',
        edad: 45
    }, {
        nombre: 'ana',
        edad: 12
    }, {
        nombre: 'luis',
        edad: 16
    }, {
        nombre: 'cristina',
        edad: 76
    }];
```

Luego el programa que recorre el array y accede a cada objeto almacenado mediante un for of:

```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>

<body>

    <script>
        const personas = [{
            nombre: 'juan',
            edad: 45
        }, {
```

Luego el programa que recorre el array y accede a cada objeto almacenado mediante un for of:

```
            nombre: 'ana',
            edad: 12
        }, {
            nombre: 'luis',
            edad: 16
        }, {
            nombre: 'cristina',
            edad: 76
        }];
        document.write('<h1>Personas mayores de edad.</h1>');
        for (let persona of personas)
            if (persona.edad >= 18)
                document.write(`${persona.nombre} y tiene ${persona.edad} años.<br>`);
    </script>
</body>
</html
```

ES6 - Estructura repetitiva 'for of'

Mediante un for of recorremos cada componente del array. En la variable persona se almacena en cada repetición del for of un elemento del array:

```
for (let persona of personas)
        if (persona.edad >= 18)
            document.write(`${persona.nombre} y tiene ${persona.edad} años.<br>`);
```

Recuperar el valor y el índice que ocupa un elemento dentro del arreglo

Imprimir todos los elementos de un array y la posición que ocupa dentro del mismo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>
<body>
    <script>
        const vector = [10, 40, 60, 5];
        for (let [indice, valor] of vector.entries())
            document.write(`${valor} ocupa la posición ${indice}<br>`);
    </script>
</body>
</html>
```

Recuperar el valor y el índice que ocupa un elemento dentro del arreglo

El método entries() devuelve un arreglo con dos valores: la posición del elemento y el valor del mismo:

```
for (let [indice, valor] of vector.entries())
```

ES6 - Map

Con la llegada de ES6 se presenta un nuevo objeto llamado 'Map' que nos evita tener que implementar desde cero una estructura de datos tipo diccionario o mapa.

La estructura de datos tipo 'Map' utiliza una clave para acceder a un valor.

ES6 - Map

Podemos relacionarlo con conceptos que conocemos:

- Un diccionario tradicional que conocemos podemos utilizar un 'Map' para representarlo. La clave sería la palabra y el valor sería la definición de dicha palabra.
- Una agenda personal también la podemos representar como un diccionario. La fecha sería la clave y las actividades de dicha fecha sería el valor.
- Un conjunto de usuarios de un sitio web podemos almacenarlo en un diccionario. El nombre de usuario sería la clave y como valor podríamos almacenar su mail, clave, fechas de login, etc.

ES6 - Map

Como clave se puede utilizar cualquier tipo de dato primitivo como también de tipo objeto. Podemos almacenar como valor también datos primitivos como objetos.

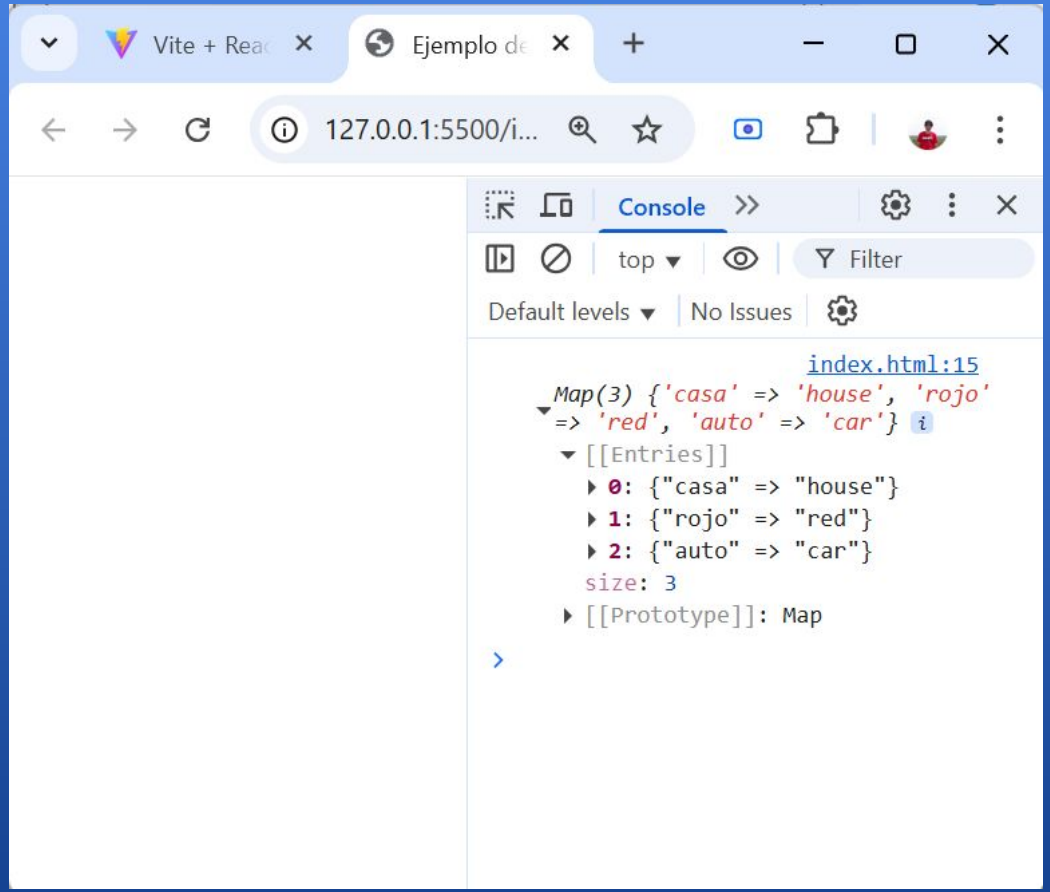
Creación de un Map

1. Podemos crear un Map vacío y posteriormente añadir elementos al mapa mediante el método 'set':

```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>

<body>
    <script>
        const mapa1 = new Map();
        mapa1.set("casa", "house");
        mapa1.set("rojo", "red");
        mapa1.set("auto", "car");
        console.log(mapa1);
    </script>
</body>

</html>
```

Creación de un Map

Utilizamos el método log del objeto console para mostrar el contenido del mapa (recordar que en Chrome y FireFox debemos presionar la tecla 'F12' para que se muestre la ventana de la console):



Creación de un Map

2. Podemos crear un Map e inmediatamente pasar los datos iniciales en el constructor del mismo:

```
<script>
        const mapa1 = new Map([
            ["casa", "house"],
            ["rojo", "red"],
            ["auto", "car"]
        ]);
        console.log(mapa1);
    </script>
```

Recuperar el valor para una determinada clave

Para recuperar un valor para una determinada clave del 'Map' disponemos del método 'get'.

Problema

Ingresar por teclado una palabra en castellano y posteriormente mostrar su traducción al inglés.

```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>
<body>
    <script>
        const mapa1 = new Map([
            ["casa", "house"],
            ["rojo", "red"],
            ["auto", "car"]
        ]);
        const palcastellano = prompt("Ingrese una palabra en castellano:");
        document.write(`La traducción de ${palcastellano} es ${mapa1.get(palcastellano)}`);
    </script>
</body>
</html>
```

Recuperar el valor para una determinada clave

Como podemos comprobar al ejecutar el algoritmo si ingresamos una palabra que existe en el 'Map' luego el método get nos retorna el valor para dicha palabra. Si ingresamos una palabra que no existe el método nos retorna el valor 'undefined'.

Disponemos de otro método en el objeto 'Map' llamado 'has' que le debemos pasar como parámetro una clave y en el caso que exista nos retorna true, en caso negativo nos retorna false. Luego podemos modificar nuestra aplicación para que nos muestre un mensaje en el caso que no exista la traducción de la palabra ingresada:

Recuperar el valor para una determinada clave

```
<script>
    const mapa1 = new Map([
    ["casa", "house"],
    ["rojo", "red"],
    ["auto", "car"]
    ]);
    const palcastellano = prompt("Ingrese una palabra en castellano:");
    if (mapa1.has(palcastellano))
    document.write(`La traducción de ${palcastellano} es ${mapa1.get(palcastellano)}`);
    else
    document.write(`No existe una traducción para la palabra ${palcastellano}`)
</script>
```

Tamaño del Map

Mediante la propiedad 'size' podemos conocer la cantidad de entradas almacenadas en el mapa:

```
<script>
    const mapa1 = new Map([
    ["casa", "house"],
    ["rojo", "red"],
    ["auto", "car"]
    ]);
    mapa1.set("ventana","window");
    console.log(mapa1.size);     // 4
</script>
```

Eliminación de elementos del Map

Podemos eliminar elementos mediante el método 'delete':

```
<script>
    const mapa1 = new Map([
    ["casa", "house"],
    ["rojo", "red"],
    ["auto", "car"]
    ]);
    mapa1.delete("casa");
    console.log(mapa1.size);     // 2
</script>
```

El método 'delete' retorna true si se eliminó la entrada en el Map y false en caso que le hayamos pasado una clave que no exista.

Eliminación de elementos del Map

Para eliminar todos los elementos del mapa disponemos del método 'clear':

```
<script>
    const mapa1 = new Map([
    ["casa", "house"],
    ["rojo", "red"],
    ["auto", "car"]
    ]);
    mapa1.clear();
    console.log(mapa1.size);    // 0
</script>
```

Recorrido de un Map

Los elementos almacenado en el 'Map' mantienen el orden en que se agregaron.

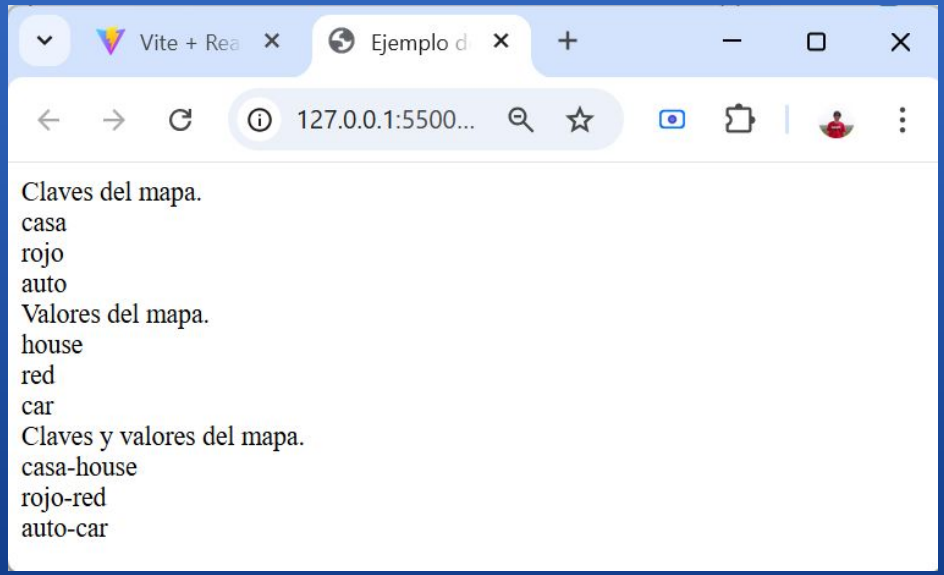
Podemos recorrer un mapa mediante la estructura repetitiva 'for of':

Recorrido de un Map

```
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>

<body>
    <script>
        const mapa1 = new Map([
            ["casa", "house"],
            ["rojo", "red"],
            ["auto", "car"]
        ]);
        document.write("Claves del mapa.<br>")
        for (let clave of mapa1.keys()) {
            document.write(clave);
            document.write("<br>");
        }
        document.write("Valores del mapa.<br>")
        for (let valor of mapa1.values()) {
            document.write(valor);
            document.write("<br>");
        }
        document.write("Claves y valores del
mapa.<br>")
        for (let [clave, valor] of mapa1) {
            document.write(clave + '-' + valor);
            document.write("<br>");
        }
    </script>
</body>

</html>
```

Recorrido de un Map

Al ejecutar el algoritmo podemos ver cómo recorrer el mapa y recuperar en forma individual solo las claves como los valores, pero también podemos recuperar en forma simultánea las claves y valores del diccionario:



Recorrido de un Map

El objeto Map igual que el objeto Array dispone de un método 'forEach' para iterar sobre los elementos del 'Map':

```
  <script>
    const mapa1 = new Map([
        ["casa", "house"],
        ["rojo", "red"],
        ["auto", "car"]
    ]);

    document.write("Claves y valores del mapa.<br>")
    mapa1.forEach((valor, clave)  => document.write(`${clave}  ${valor}<br>`));
</script>
```

EJERCICIOS ADICIONALES PROPUESTOS

¡MUCHAS GRACIAS!