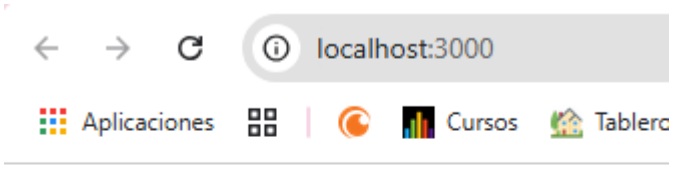
Dossier React

NOTA: Tengo hasta la práctica 46, pero no me dio tiempo de meterlo todo en el pdf.

Práctica 1

Crear el hola mundo descrito y agrega tu nombre completo al (usando npx para crear la app y npm start para arrancarla como se indica en el tema)

```
import React from 'react'

type Props = {}

const Practica1 = (props: Props) => {
    return (
        <div className="Practica1">
        <h3> Hola Mundo! Soy Melissa</h3>
        </div>
        );

}

export default Practica1
```

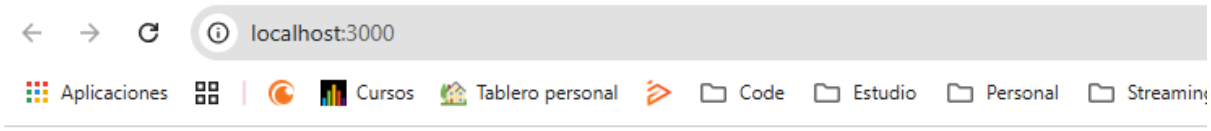


Hola Mundo! Soy Melissa

Práctica 2

Realizar lo descrito y tomar captura de pantalla del mensaje en el navegador (hay que recordar que por defecto la web está en el puerto 3000)

```
import React from 'react';
import ReactDOM from 'react-dom';

const mensaje = <h1>Vamos a renderizar este mensaje en nuestra web</h1>;
const divRoot = document.getElementById("root");
ReactDOM.render( mensaje, divRoot);
```

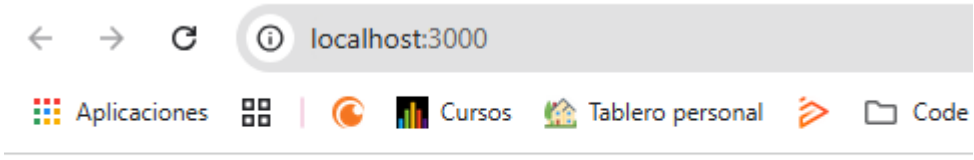


Vamos a renderizar este mensaje en nuestra web

Práctica 3

Reproducir el ejemplo anterior, pero en lugar de mostrar números primos en el dirá: "mis datos:" y en el h4 le habremos pasado un objeto literal JSON con tu nombre, apellidos y estudios que estás realizando

```
const ComponenteApp = () => {
    const misDatos = {
        nombre: "Melissa",
        edad: 27,
        ciudad: "La Orotava"
    };
return (
    <>
        <h1>Mis datos:</h1>
        <h4>{JSON.stringify(misDatos)}</h4>
    </>

);
}
export default ComponenteApp;
```



Mis datos:

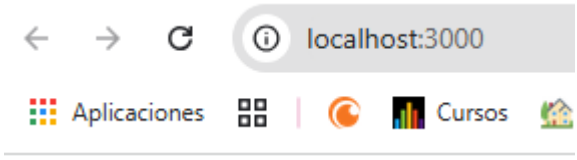
{"nombre":"Melissa","edad":27,"ciudad":"La Orotava"}

Práctica 4

Reproducir el ejemplo anterior, pero cambiando que los atributos que reciba sean: num1 y num2 y lo que muestre es: La suma de num1 y num2 es: num1 + num2 (donde num1 y num2 serían los datos que recibiera el componente)

```
import React from "react";
import PropTypes from 'prop-types';
const ComponenteApp = (props) => {

    return (
        <>

            <p>La suma de num1 y num2 es: {props.num1 + props.num2}</p>
        </>
    );
};

// Validación de tipos de propiedades
ComponenteApp.propTypes = {
    num1: PropTypes.number.isRequired,
    num2: PropTypes.number.isRequired
};

export default ComponenteApp;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import ComponenteApp from './ComponenteApp';

const divRoot = document.getElementById("root");
ReactDOM.render(<ComponenteApp num1={5} num2={10} />, divRoot);
```



La suma de num1 y num2 es: 15

Práctica 6

Conseguir el renderizado anterior, generando el componente Reloj.ts apropiado. Para ello generaremos el fichero: Reloj.ts y dentro estará el componente TSX

Reloj.ts

```
import React from 'react';

type Props = {
  zona?: string;
};

export const Reloj = ({ zona }: Props): React.ReactElement => {
  const zonaString = zona ?? "Europe/Madrid";
  const fecha = new Date().toLocaleString("es-ES", { timeZone: zonaString });
// Obtiene la fecha en la zona horaria especificada



  return React.createElement(
    'div',
    null,
    React.createElement('h2', null, `Hora en ${zonaString}:`),
    React.createElement('p', null, fecha)
  );
};
```

RelojesMundiales.tsx

```tsx
import React from 'react';
import { Reloj } from './Reloj';

type Props = {};

export const RelojesMundiales = (props: Props) => {
  return (
    <div>
      <h1>Actividad React: Relojes Mundiales</h1>
      <Reloj zona="Europe/Madrid" />
      <Reloj zona="America/New_York" />
      <Reloj zona="Europe/London" />
    </div>
  );
};

export default RelojesMundiales;
```

Actividad React: Relojes Mundiales

Hora en Europe/Madrid:

7/11/2024, 17:37:38

Hora en America/New_York:

7/11/2024, 11:37:38

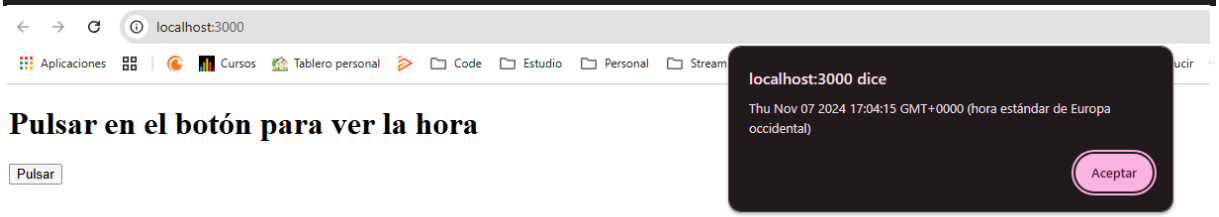
Hora en Europe/London:

7/11/2024, 16:37:38

Práctica 7

Probar el código anterior. Tomar captura de pantalla del navegador al pulsar el botón

```
type Props = {}

const Practica7 = (props: any) => {

    const mostrarHora = ()=>{
        alert(new Date());
    }

    return (
    <>
    <h1> Pulsar en el botón para ver la hora</h1>
    <button onClick={mostrarHora}>Pulsar</button>
    </>
    );

}

export default Practica7
```

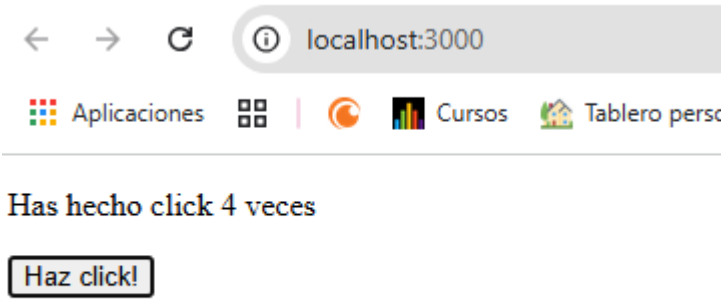


Práctica 8

Crear el código anterior de componente Contador en un fichero nuevo y cargar en index.tsx en la parte de renderizado: ReactDOM.render() ese componente Probarlo en el navegador y comprobar que efectivamente cambia el contador con los click

```
import React, { Component } from 'react'
class Contador extends Component {
    state = { count: 0 } // inicializamos el state a 0
    render () {
        const { count } = this.state // extraemos el count del state
        return (
            <div>
                <p>Has hecho click {count} veces</p>
                { /* Actualizamos el state usando el método setState */ }
                <button onClick={() => this.setState({ count: count + 1 })}>
```

```
                Haz click!
                </button>
            </div>
        )
    }
}
export default Contador;
```



Práctica 9

Realizar con el Hook useState dentro de un functional component un componente que sirva a un usuario para prácticar la tabla del 2. Cada vez que pulse en el botón se le mostrará la solución correcta de la tabla. Así: la primera vez que haga clic se le mostrará: 2x1 = 2 La segunda vez: 2x2=4 y así sucesivamente. En definitiva: que vaya mostrando la tabla del 2 a cada click Observar que después de 2x10 mostrará 2x1

```
const Practica9 = (props: Props) => {
  const [Contador, setContador] = useState(1)

  const manejarClick = () => {
    setContador(number => (number === 10 ? 1 : number + 1)) //incrementar
contador pero si llea a 1 reiniciar a 1
  };
  return (
    <>
      <h2>Tabla del 2</h2>
      <button onClick={manejarClick}>
        <p>2 x {Contador} = {2 * Contador}</p>
      </button>
    </>
  )
}
```

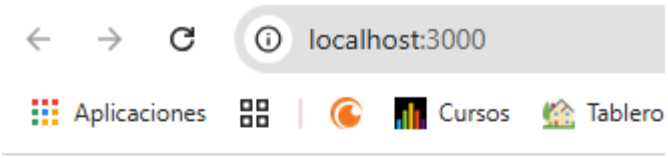


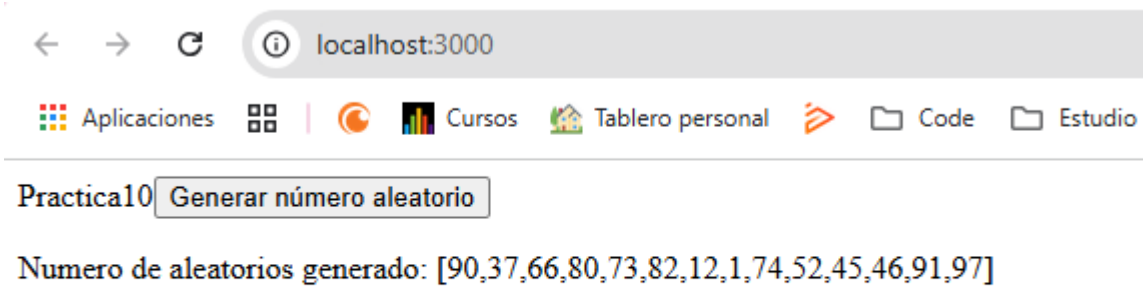
Tabla del 2

2 x 3 = 6

Práctica 10

Crear un functional component react (usa el snippet: tsrafc) que tenga un botón. Este botón al pulsarlo va agregando un nuevo número aleatorio de 0 a 100 de tal forma que podemos ver gracias al state toda la lista de aleatorios generados (Nota: podemos usar: JSON.stringfy(nombredelarray) para ver el array u otro objeto) Nota: hay una forma sencilla de crear un nuevo array con un nuevo elemento conservando los datos del anterior. Imaginemos que queremos agregar el número 5: const arrayanterior: Array = [4, 2, 7]; [...arrayanterior, 5]

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

type Props = {}

const Practica10 = (props: Props) => {
    const  [aleatorio, setaleatorio] = useState<Array<Number>>([]);

    const manejarClick = () => {
        setaleatorio([...aleatorio,Math.floor((Math.random() * 100)+1)]);
    };
  return (
    <div>Practica10
        <button onClick={manejarClick}>Generar número aleatorio</button>
        <p>Numero de aleatorios generado: {JSON.stringify(aleatorio)}</p>
    </div>
  )
}

export default Practica10
```

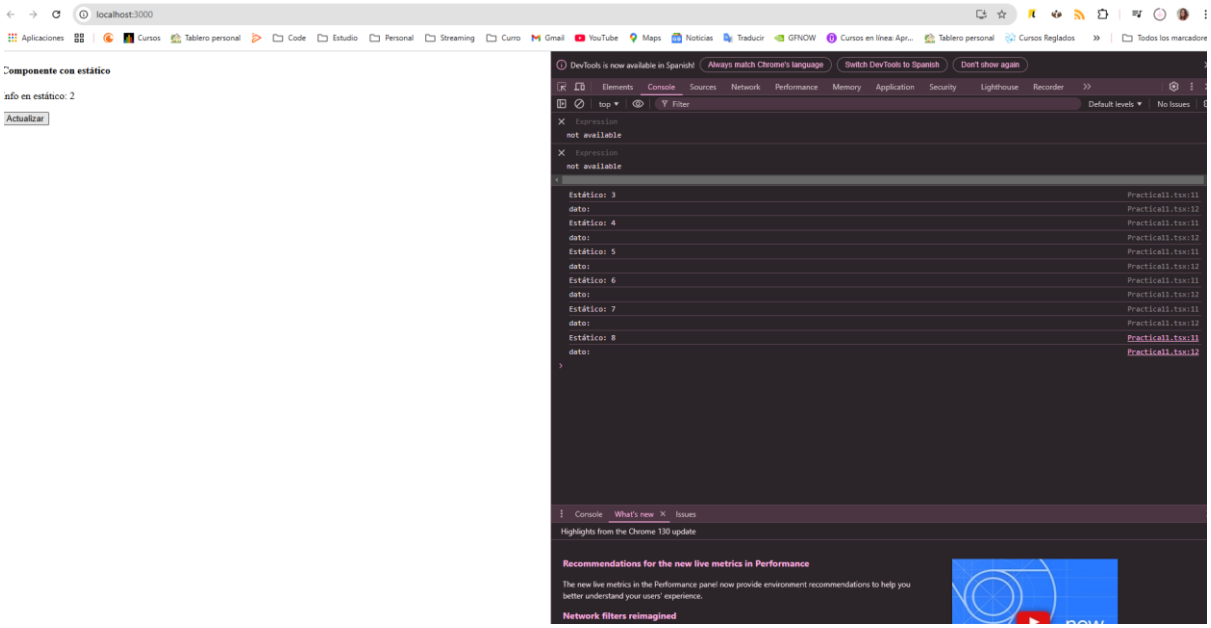


Práctica 11

Crear el anterior functional component, ejecútalo y abre la consola ¿ se está actualizando la información del atributo estático ? ¿ y de la variable: dato ? Ahora quita el comentario de la línea: sethoraactual(“” + new Date()); Sabemos que de esa manera al actualizar el state se fuerza un nuevo renderizado ¿ se está actualizando la info del atributo estático ? ¿ y de la variable: dato ?

```
const Practica11 = () => {
    const [horaactual, sethoraactual] = useState("");
    let dato = 1;

    function actualizar() {
        Practica11.atributoEstatico++;
        dato++;
        console.log("Estático: " + Practica11.atributoEstatico);
        console.log("dato: ")
        //sethoraactual("" + new Date());
    }

  return (
    <div>
        <h4>Componente con estático</h4>
        <p>Info en estático: {Practica11.atributoEstatico}</p>
        <button onClick={actualizar}>Actualizar</button>

    </div>
  )
}

Practica11.atributoEstatico = 2;
```

Como podemos observar no se actualiza en el renderizado al ejecutarse la funcion de actualizar, en consola se está cambiando pero no renderiza:



Al quitarle los comentarios al set, se renderiza todo correctamente

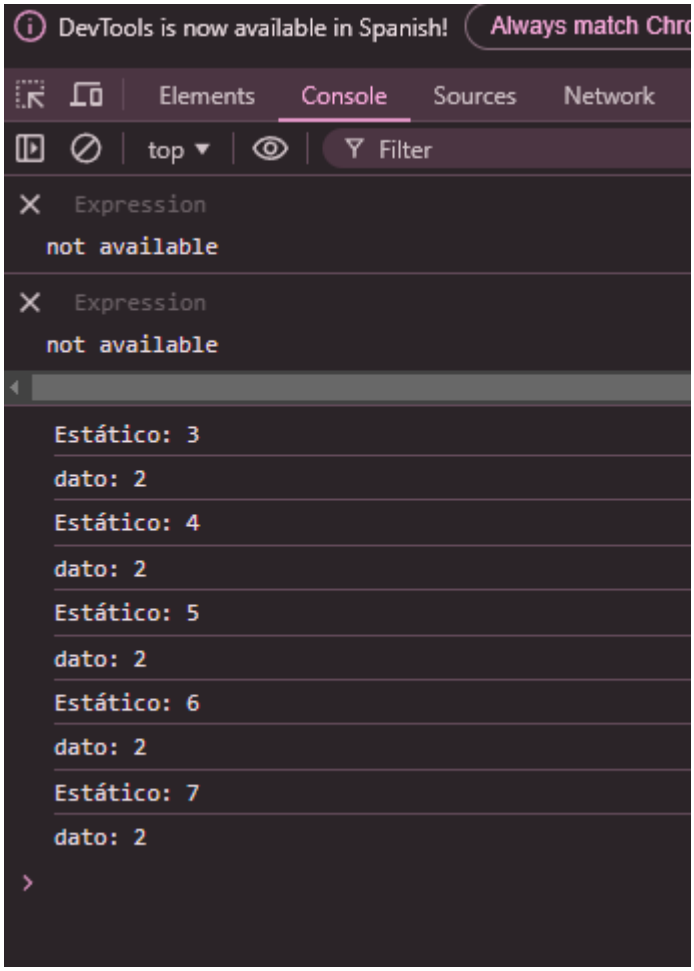
`Practica11.atributoEstatico` (atributo estático):

`Practica11.atributoEstatico` es una propiedad de la función `Practica11` (es decir, un "atributo estático" fuera del ciclo de renderizado de React).

- **Debido a que no está relacionado con el estado de React, siempre retiene su valor entre renderizados.**
- **Como el valor de `Practica11.atributoEstatico` se está incrementando directamente (sin el uso de `useState`), se mantiene correctamente actualizado y puedes verlo en la consola después de cada clic.**

`dato` (variable local):

- **`dato` es solo una variable local de la función. Cada vez que el componente se vuelve a renderizar (lo cual ocurre cuando usas `sethoraactual` para actualizar el estado), `dato` se reinicia.**
- **Esto sucede porque cada renderizado de un componente funcional en React se ejecuta en una nueva instancia de la función, por lo que todas las variables definidas dentro de la función (como `dato`) se restablecen a su valor inicial.**
- **Si incrementas `dato` dentro de la función `actualizar()`, solo se incrementa por ese ciclo de ejecución y luego se pierde cuando el componente se vuelve a renderizar.**

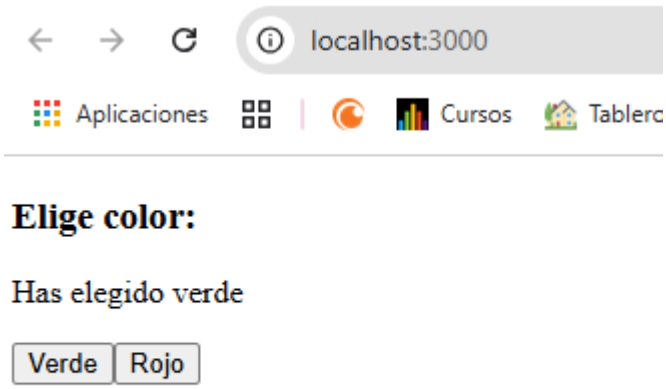


Práctica 12

Crear la actividad que se acaba de describir. Notar que hay que usar un useState para que muestre un texto u otro según lo que se haya pulsado

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

type Props = {}

const Practica12 = (props: Props) => {
  const [color, setColor] = useState("")
  function elegirColor(color: string) {
    setColor(color)

  }
  return (
    <>
    <h3>Elige color:</h3>
    <p>Has elegido {color}</p>
    <button onClick={()=>elegirColor("verde")}>Verde </button>
    <button onClick={()=>elegirColor("rojo")}>Rojo</button>
    </>
```

```
    )
}

export default Practica12
```



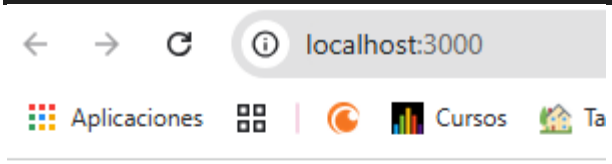
Práctica 13

Reproducir el ejemplo anterior en la aplicación monedas. Hacer que los li no muestren únicamente el nombre de la moneda sino también el país. Ej:

- **libra de UK**

```
const Practica13 = (props: Props) => {
    const [monedas, setmonedas] = useState<Array<Moneda>>([]);

    function addMoneda(){
        const moneda: Moneda ={
            nombre: "libra",
            pais: "uk"
        }
        setmonedas([...monedas,moneda]);
    }

    return (
    <>
    <h3>Cliente de monedas</h3>
    <div>
        <button onClick={addMoneda}>
        agregar moneda
        </button>
        <h4>Monedas: </h4>
```

```
        <ul>
        {
        monedas.map( (m:Moneda) => {
         return (
            <li> Moneda: {m.nombre}, Pais: {m.pais} </li>
            );
            })
        }
        </ul>
    </div>
</>
    );
}
```



Cliente de monedas

agregar moneda

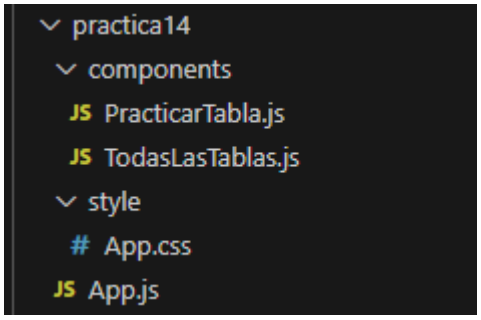
Monedas:

- libra de uk
- libra de uk
- libra de uk
- libra de uk
- libra de uk

Practica 14

Crear un componente: TodasLasTablas que use el componente ya creado Así muestra las tablas del 2 al 10 (mirar imagen ejemplo) Se usarán las pros: → Esto genera la tabla del 5. Usar un map para un array [2,3,..,10] y establece para cada componente PracticarTabla el prop para su tabla

Estructuramos tal que así el ejercicio:



Componente App.js

En tu archivo App.js, importa y renderiza el componente TodasLasTablas para mostrar las tablas.

```
const App = () => {
  return (
    <div className="App">
      <h1>Tablas de Multiplicar</h1>
      <TodasLasTablas />
    </div>
  );
}
```

Componente PracticarTabla.js

Este es el componente que hicimos en la practica 9 para practicar la tabla:

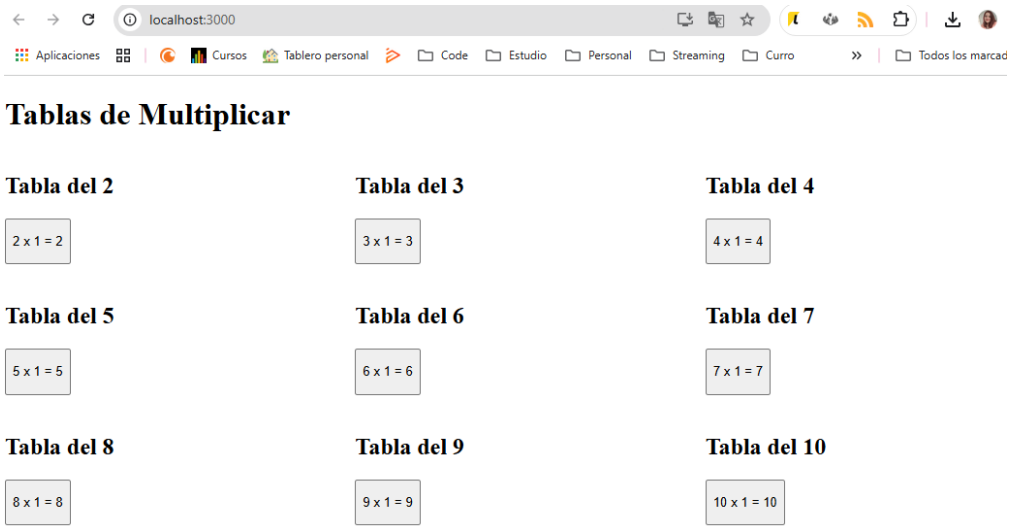
```
interface Props {
  tabla: number;
}

const PracticarTabla: React.FC<Props> = ({ tabla }) => {
  const [contador, setContador] = useState(1);

  const manejarClick = () => {
    setContador(prevContador => (prevContador === 10 ? 1 : prevContador + 1));
  };

  return (
    <div>
      <h2>Tabla del {tabla}</h2>
      <button onClick={manejarClick}>
        <p>{tabla} x {contador} = {tabla * contador}</p>
      </button>
    </div>
  );
};
```

```
export default PracticarTabla;
```

Componente TodasLasTablas.js

Este es el componente que generará las tablas del 2 al 10, pasando cada número como prop tabla al componente PracticarTabla.

```
const TodasLasTablas: React.FC = () => {
  const arr = [2, 3, 4, 5, 6, 7, 8, 9, 10];

  return (
    <div className="grid">
      {arr.map((tabla, index) => (
        <PracticarTabla key={index} tabla={tabla} />
      ))}
    </div>
  );
};

export default TodasLasTablas;
```



Práctica 15

Crear el renderizado anterior. Al pulsar en botón rojo el área tiene color fuente rojo y borde rojo. Si se pulsa en verde, pues en verde, y así con todos. Se recomienda crear las 4 clases CSS y luego que se establezcan mediante:

Este area muestra los resultados de los botones

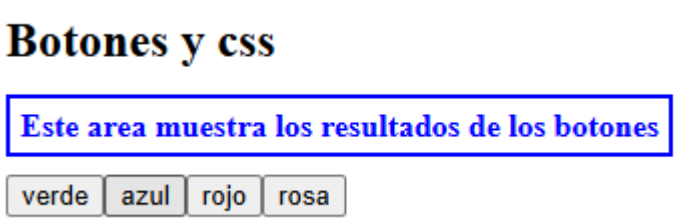
Creo la hija de estilos con cada clase, verde rojo azul y rosa

```css
.verde {
    color: green;
    border-color: green;
    border: 2px solid;
    padding: 5px;
    display: inline;
}

.azul {
    color: blue;
    border-color: blue;
    border: 2px solid;
    padding: 5px;
    display: inline;
}

.rojo {
    color: red;
    border-color: red;
    border: 2px solid;
    padding: 5px;
    display: inline;
}

.rosa {
    color: pink;
    border-color: pink;
    border: 2px solid;
    padding: 5px;
    display: inline;
}

button {
    margin-top: 15px;
}
```

Creamos el componente, y metemos onclick en todos los botones que cambie el estado de la clase que vamos a aplicar en el css

```
const Practica15 = () => {

    const [claseaplicada, setClaseAplicada] = useState("");

  return (
    <div>
        <h2>Botones y css</h2>
        <h4 className={claseaplicada}>Este area muestra los resultados de los
botones</h4>
        <div>
            <button onClick={() => setClaseAplicada("verde")}>verde</button>
            <button onClick={() => setClaseAplicada("azul")}>azul</button>
            <button onClick={() => setClaseAplicada("rojo")}>rojo</button>
            <button onClick={() => setClaseAplicada("rosa")}>rosa</button>
        </div>

    </div>
  )
}
```

Y listo:



Práctica 16

En la práctica de los relojes de zonas horarias, crear un array con 5 zonas horarias, entre ellas: Londres, Madrid y usando array.map generar los 5 componentes Reloj con su respectiva propiedad timezone, dándole estilos CSS a los componentes

Reloj.tsx

```
interface Props {
  zona: string;
}

export const Reloj = ({ zona }: Props): React.ReactElement => {
  const zonaString = zona ?? "Europe/Madrid";
  const [fecha, setFecha] = useState<string>(new Date().toLocaleString("es-
ES", { timeZone: zonaString }));

  useEffect(() => {
    const interval = setInterval(() => {
      setFecha(new Date().toLocaleString("es-ES", { timeZone: zonaString }));
    }, 1000);

    return () => clearInterval(interval);
  }, [zonaString]);

  return (
    <div>
      <h2>Hora en {zonaString}:</h2>
      <p>{fecha}</p>
    </div>
  );
};
```

RelojesMundiales.tsx

```
export const RelojesMundiales = (props: Props) => {
  const zonas = [
    "Europe/Madrid",
    "America/New_York",
    "Europe/London",
    "Asia/Tokyo",
    "Australia/Sydney",
  ];

  return (
    <div className="container">
      <h1>Actividad React: Relojes Mundiales</h1>
```

```
      <div className="relojes-container">
        {zonas.map((zona, index) => (
          <Reloj key={index} zona={zona} />
        ))}
      </div>
    </div>
  );
};

export default RelojesMundiales;
```

Actividad React: Relojes Mundiales

Hora en Europe/Madrid:

15/11/2024, 12:50:39

Hora en America/New_York:

15/11/2024, 6:50:39

Hora en Europe/London:

15/11/2024, 11:50:39

Hora en Asia/Tokyo:

15/11/2024, 20:50:39

Hora en Australia/Sydney:

15/11/2024, 22:50:39

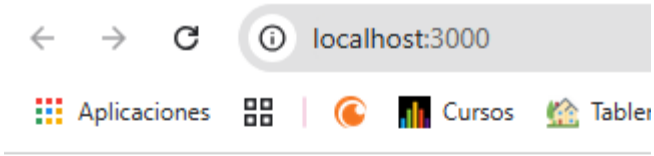
Práctica 17:

Crear un componente que tenga dos botones. Cuando se pulse en el primer botón se cargará un componente que mostrará 10 números aleatorios de 0 a 100 a pulsar un botón llamado "generar" que esté dentro del componente Si se pulsa en el otro botón se carga otro componente que reemplaza el anterior que muestra un saludo y la fecha actual (la fecha se enviará mediante props)

```
const NúmerosAleatorios = () => {
    const [numeros, setNumeros] = useState<number[]>([]);

    const generarNumeros = () => {
      const nuevosNumeros = Array.from({ length: 10 }, () =>
Math.floor(Math.random() * 101));
      setNumeros(nuevosNumeros);
    };

    return (
      <div>
        <button onClick={generarNumeros}>Generar</button>
        <ul>
          {numeros.map((numero, index) => (
            <li key={index}>{numero}</li>
          ))}
        </ul>
      </div>
    );
  };

  export default NúmerosAleatorios;
```

```
const Saludo = ({ fecha }: Props) => {

  return (
    <div>
      <h2>¡Hola! Aquí está la fecha actual:</h2>
      <p>{fecha}</p>
    </div>
  )
}

export default Saludo
```

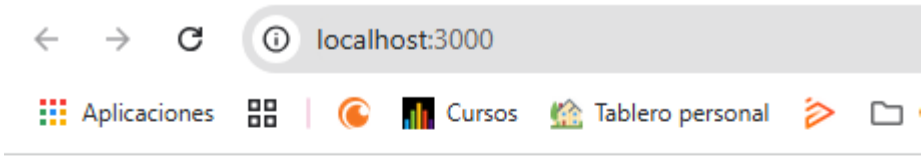
```
const App = () => {

    const [mostrarNumeros, setMostrarNumeros] = useState<boolean>(true);
    const fechaActual = new Date().toLocaleString();

    return (
      <div>
        <button onClick={() => setMostrarNumeros(true)}>Generar
Números</button>
        <button onClick={() => setMostrarNumeros(false)}>Mostrar
Saludo</button>

        {/* Condicional para renderizar uno u otro componente */}
        {mostrarNumeros ? (
          <NumerosAleatorios />
        ) : (
          <Saludo fecha={fechaActual} />
        )}
      </div>
    );
  };

  export default App;
```



Generar Números | Mostrar Saludo

Generar

- 3
- 46
- 28
- 46
- 39
- 65
- 61
- 82
- 88
- 76



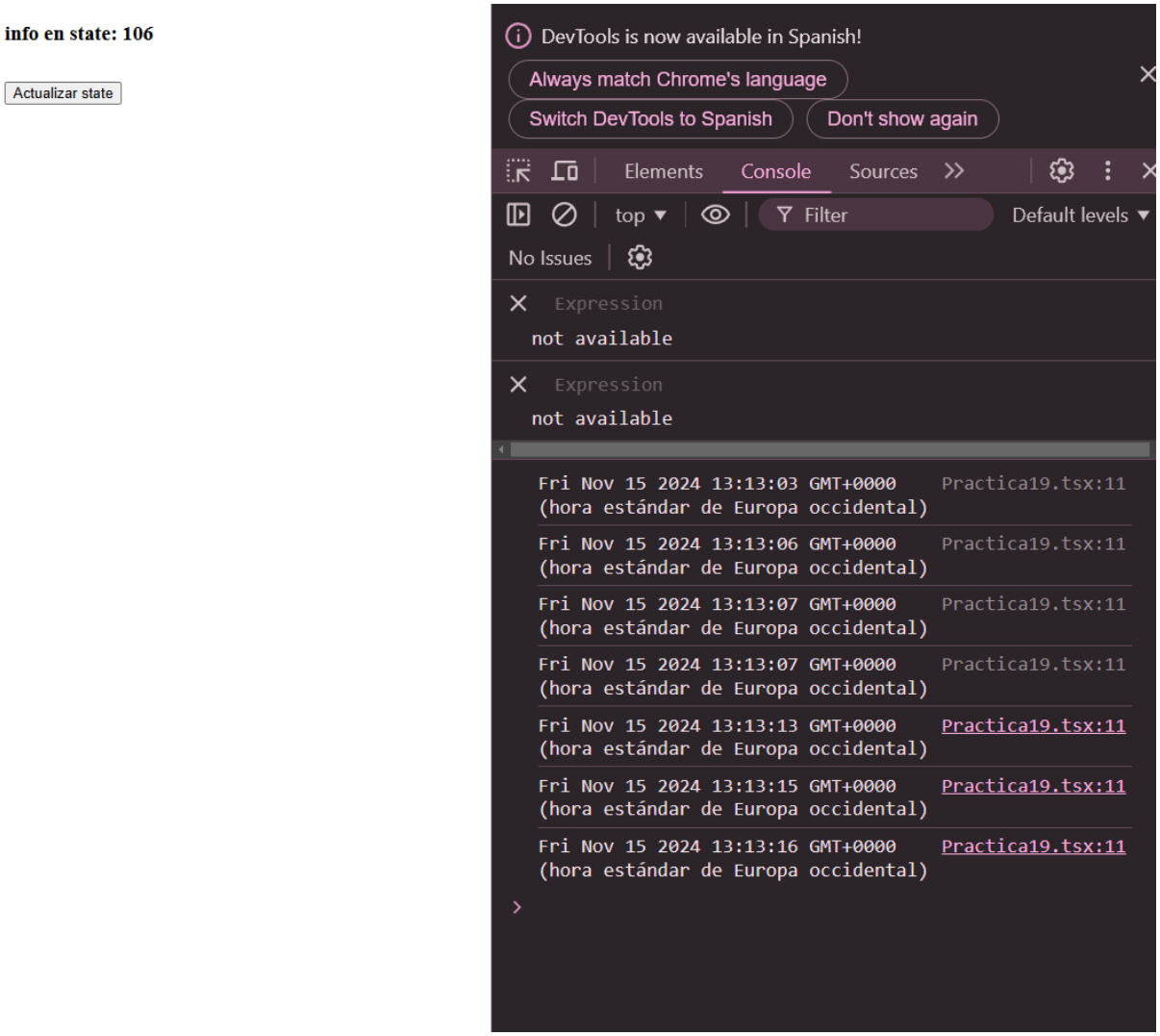
Generar Números | Mostrar Saludo

¡Hola! Aquí está la fecha actual:

15/11/2024, 13:03:01

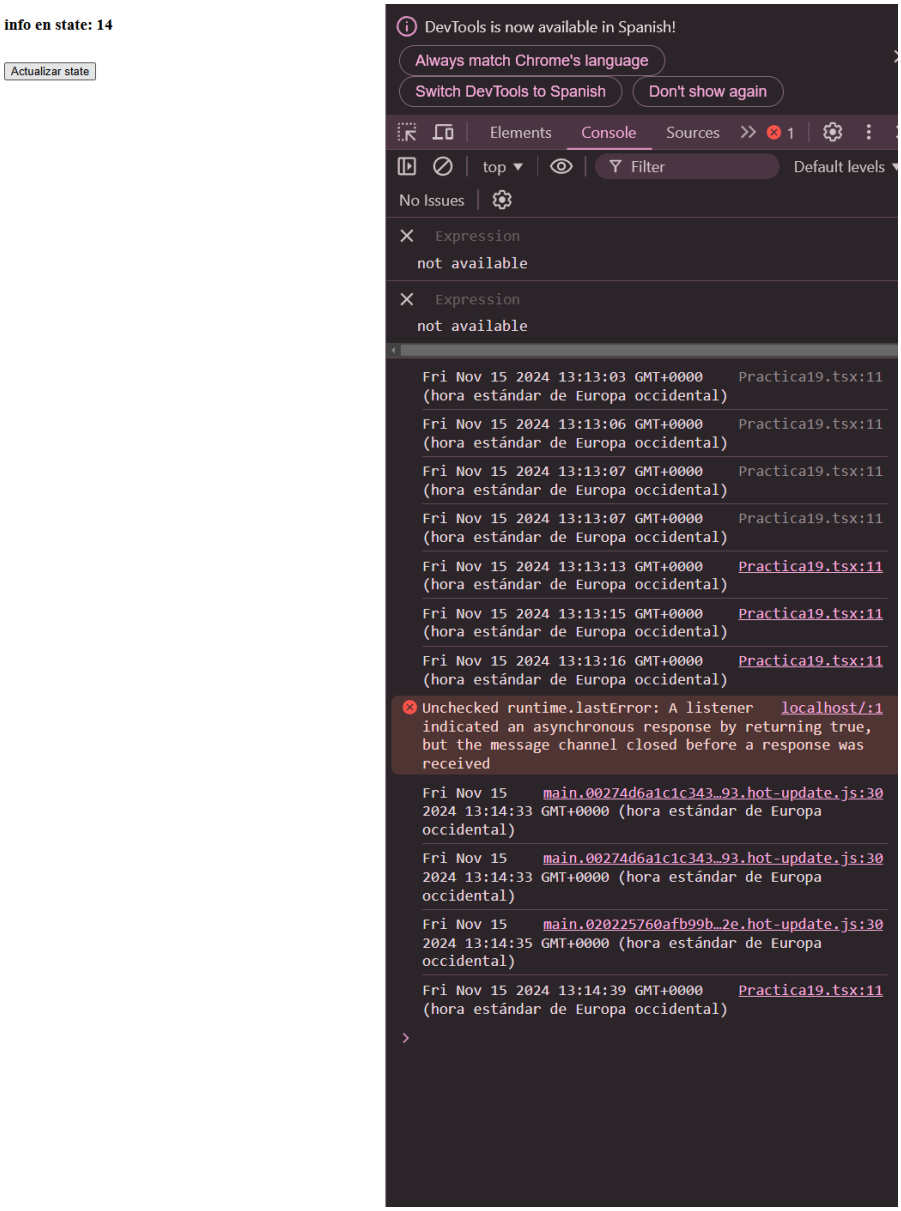
Práctica 19

Abriendo la consola para ver los mensajes de log, ejecutar el código anterior. ¿ se muestra la fecha cada vez que se renderiza (modifica el estado) ? ¿ el contador empieza en qué número ? Ahora modifica el código anterior quitando los comentarios en la línea: //setContador(-1) ¿ qué ocurre ahora ? ¿ En el primer renderizado (antes de pulsar el botón) qué muestra el contador? ¿Y después de ejecutar el botón? Sigue modificando el código quitando los comentarios en el array de useEffect quedando la línea final del useEffect() así: }, []) ¿ se ejecuta es useEffect() en cada renderizado ? ¿ se ejecuta en el momento del montaje ? Finalmente vamos a dejar nuestro useEffect así: useEffect(() => { const efecto = () =>{ let fecha = new Date(); console.log(fecha); setcontador(-1); } efecto(); }, [contador>10]) Ahora ¿cuándo se ejecuta el useEffect?



No, la fecha solo se muestra en el primer renderizado. Como el array de dependencias `[]` está vacío, el efecto solo se ejecuta una vez, cuando el componente se monta. Es decir, el efecto no se vuelve a ejecutar cada vez que el estado `contador` cambia. El contador comienza con el valor de `100`, ya que eso es lo que se le pasa inicialmente al hook `useState()`.

Al descomentar esto es lo que ocurre:



En el primer renderizado, cuando el componente se monta, el `useEffect()` se ejecuta y muestra la fecha en la consola, pero además ejecuta `setContador(-1)`, lo cual cambia el estado de `contador` a `-1`. Esto disparará un nuevo renderizado debido al cambio de estado.

Después del primer renderizado, el estado de `contador` cambia a `-1`, y el componente se vuelve a renderizar con el valor actualizado. Sin embargo, el `useEffect` no se ejecutará nuevamente, ya que el array de dependencias está vacío (`[]`), lo que significa que solo se ejecuta una vez.

El contador muestra el valor `100` en el primer renderizado. Luego, el `useEffect()` cambia el valor a `-1` y el componente se vuelve a renderizar con ese valor.

Al hacer clic en el botón, se actualiza el estado y el contador aumenta en `1`, es decir, pasará de `-1` a `0`.

Práctica 20

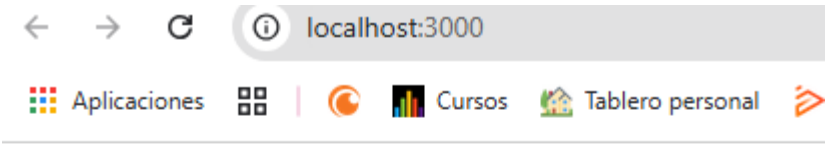
Realizar un componente para el juego de Acertar número secreto (de 0 a 9). Tendremos 10 botones siguiendo el patrón: apostar(7)}" > 7 Al montarse el componente se genera el número aleatorio secreto, que permanecerá sin modificación hasta que el usuario acierte el número. Cuando se pulsa en los botones de apuesta se informa al usuario de si ha acertado, si el número es menor o mayor que secreto

Plan:

1. **Generar el número secreto cuando el componente se monte. Usaremos `useEffect` para este comportamiento.**
2. **Mantener el estado del número secreto. El número no cambiará hasta que el usuario lo adivine.**
3. **Configurar los botones que representarán las apuestas del número.**
4. **Informar al usuario si la apuesta fue correcta, o si el número es mayor o menor que el secreto.**

```
const Memorize = () => {
    // Estado para almacenar el número secreto
    const [numeroSecreto, setNumeroSecreto] = useState<number | null>(null);
    // Estado para almacenar el mensaje de resultado (si acertó, mayor o
menor)
    const [mensaje, setMensaje] = useState<string>('');

    // Se genera el número secreto cuando el componente se monta
    useEffect(() => {
      const numeroAleatorio = Math.floor(Math.random() * 10); // Número entre
0 y 9
      setNumeroSecreto(numeroAleatorio);
    }, []); // El efecto solo se ejecuta una vez al montar el componente

    // Función que se ejecuta cuando el usuario hace una apuesta
    const apostar = (numeroApostado: number) => {
      if (numeroSecreto !== null) {
        if (numeroApostado === numeroSecreto) {
          setMensaje('¡Felicidades, acertaste!');
        } else if (numeroApostado < numeroSecreto) {
          setMensaje('El número secreto es mayor. ¡Inténtalo de nuevo!');
        } else {
          setMensaje('El número secreto es menor. ¡Inténtalo de nuevo!');
        }
      }
    };
```

```
  return (
    <div>
      <h2>Juego de Acertar el Número Secreto</h2>
      <p>{mensaje}</p>
      <div>
        {[...Array(10)].map((_, index) => (
          <button key={index} onClick={() => apostar(index)}>
            {index}
          </button>
        ))}
      </div>
    </div>
  );
};
```



Juego de Acertar el Número Secreto

¡Felicidades, acertaste!

0 1 2 3 4 5 6 7 8 9

Práctica 21

Copiar y ejecutar el ejemplo anterior. Buscar información sobre setInterval() ¿ qué significa el 1000 que le pasamos como parámetro ? ¿ para qué vale el valor devuelto timerID ?. Comentar la línea: setfechaactual(newfecha) de la función tick() y escribir en su lugar: console.log(newfecha); ¿ qué ocurre con el renderizado ? Mirar en la consola que información está mostrando y explicar lo que ocurre

Explicación de setInterval() y el parámetro 1000:

javascript

Copiar código

```
const timerID = setInterval(tick, 1000);
```

setInterval() es una función en JavaScript que ejecuta una función o fragmento de código repetidamente en intervalos de tiempo fijos. En este caso, se ejecuta la función tick cada 1000 milisegundos (1 segundo).

El parámetro 1000 indica el intervalo de tiempo, en milisegundos, entre cada ejecución de la función tick. Es decir, la función tick se ejecutará una vez cada segundo.

Valor devuelto por setInterval() (timerID): setInterval() devuelve un identificador único (timerID) que se puede usar para detener el intervalo con clearInterval(timerID) si lo necesitas más adelante. Si no necesitas detener el intervalo, este valor no es obligatorio para el funcionamiento del código.

2. Qué sucede si comentamos setfechaactual(newfecha) y usamos console.log(newfecha):

```
function tick() {
  const newfecha = "" + new Date();
  // setfechaactual(newfecha);  // Comentado
  console.log(newfecha);  // Reemplazado con log
}
```

Cuando reemplazamos setfechaactual(newfecha) con console.log(newfecha), lo que estamos haciendo es:

Renderizado: La función tick se sigue ejecutando cada segundo, pero ya no se actualiza el estado de fechaactual. Esto significa que React ya no volverá a renderizar el componente porque no hay un cambio en el estado (fechaactual).

Consola: Ahora, cada vez que la función tick se ejecute (cada segundo), se imprimirá la fecha y hora actual en la consola, pero el componente no se vuelve a renderizar en la interfaz de usuario.

Explicación de lo que ocurre:

Sin setfechaactual(newfecha): Solo se registra el valor de la fecha en la consola cada segundo, pero como no se cambia el estado del componente, no hay un nuevo renderizado en la interfaz de usuario. Es decir, solo verás los logs en la consola pero no habrá actualizaciones visuales en el navegador.

Con setfechaactual(newfecha): Cada segundo, la fecha se guarda en el estado del componente y React volverá a renderizar el componente con el nuevo valor de fechaactual. La interfaz de usuario mostrará la hora actualizada cada segundo.

Práctica 22

Ahora que ya sabemos usar setInterval() y combinarlo con useEffect() modificar la actividad de los relojes mundiales de tal forma que se muestren con la información de la hora actualizada cada segundo

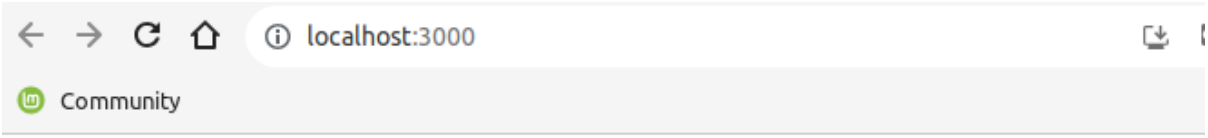
```
export const Reloj = (props: Props) => {
const [fecha, setFecha] = useState<string>("");

useEffect(() => {
const timerID = setInterval(tick, 1000);
return () => clearInterval(timerID);
}, []);

const tick = () => {
const zona = props.zona || "Europe/Madrid";
const nuevaFecha = new Date().toLocaleString("es-ES", { timeZone: zona });
setFecha(nuevaFecha);
};

return (
<div>
<h2>Hora en {props.zona || "Europe/Madrid"}:</h2>
<p>{fecha}</p>
</div>
);
};
```

Práctica 23

Usando useRef(), crear un componente con 2 input y un párrafo (etiqueta:

<p>) donde uno de los inputs sea para el nombre y el otro input para los apellidos. Al pulsar en el botón tomará la información de los dos inputs y lo mostrará en el párrafo concatenados y dirá cuántas letras tiene el nombre completo

```
const Practica23 = () => {
const inputnombre = useRef<HTMLInputElement>({} as HTMLInputElement);
const inputapellido = useRef<HTMLInputElement>({} as HTMLInputElement);
const divresultado = useRef<HTMLDivElement>({} as HTMLDivElement);

function establecerLength() {
let htmlinputnombre = inputnombre.current;
let nombre = String(htmlinputnombre.value);
let htmlinputapellido = inputapellido.current;
let apellido = String(htmlinputapellido.value);

let htmldiv = divresultado.current;
let longitud = nombre.length + apellido.length;

htmldiv.innerText = `La longitud de tu nombre y apellido es: ${longitud}`;
}
return (
<div>
<h2>Introduzca su nombre y apellido y calculamos la longitud</h2>
<input type="text" ref={inputnombre} placeholder="Nombre" />
<input type="text" ref={inputapellido} placeholder="Apellido" />
<button onClick={establecerLength}>Calcular longitud</button>
<div ref={divresultado}></div>

</div>
)
}
```



Introduzca su nombre y apellido y calculamos la longitud

Melissa Ruiz Calcular longitud

La longitud de tu nombre y apellido es: 11

Práctica 24

Modificar el ejercicio de acertar número. Ahora en lugar de 10 botones, habrá un único input y un único botón. Al pulsar el botón en la acción que desencadene se usará useRef() para tomar la información que haya en el input y así realizar la apuesta

```
const Practica24 = () => {
// Estado para almacenar el número secreto
const [numeroSecreto, setNumeroSecreto] = useState<number | null>(null);
// Estado para almacenar el mensaje de resultado (si acertó, mayor o menor)
const [mensaje, setMensaje] = useState<string>('');
// Referencia para capturar el valor del input
const inputRef = useRef<HTMLInputElement>(null);

const generarNuevoNumeroSecreto = () => {
const numeroAleatorio = Math.floor(Math.random() * 10); // Número entre 0 y 9
setNumeroSecreto(numeroAleatorio);
setMensaje('');
if (inputRef.current) {
inputRef.current.value = ''; // Limpiar el input
}
};


// Función que se ejecuta cuando el usuario hace una apuesta
const apostar = () => {
const valorInput = inputRef.current?.value;
if (valorInput && numeroSecreto !== null) {
const numeroApostado = parseInt(valorInput, 10);
if (numeroApostado === numeroSecreto) {
setMensaje('¡Felicidades, acertaste!');
} else if (numeroApostado < numeroSecreto) {
setMensaje('El número secreto es mayor. ¡Inténtalo de nuevo!');
} else {
setMensaje('El número secreto es menor. ¡Inténtalo de nuevo!');
}
}
};

return (
<div>
```

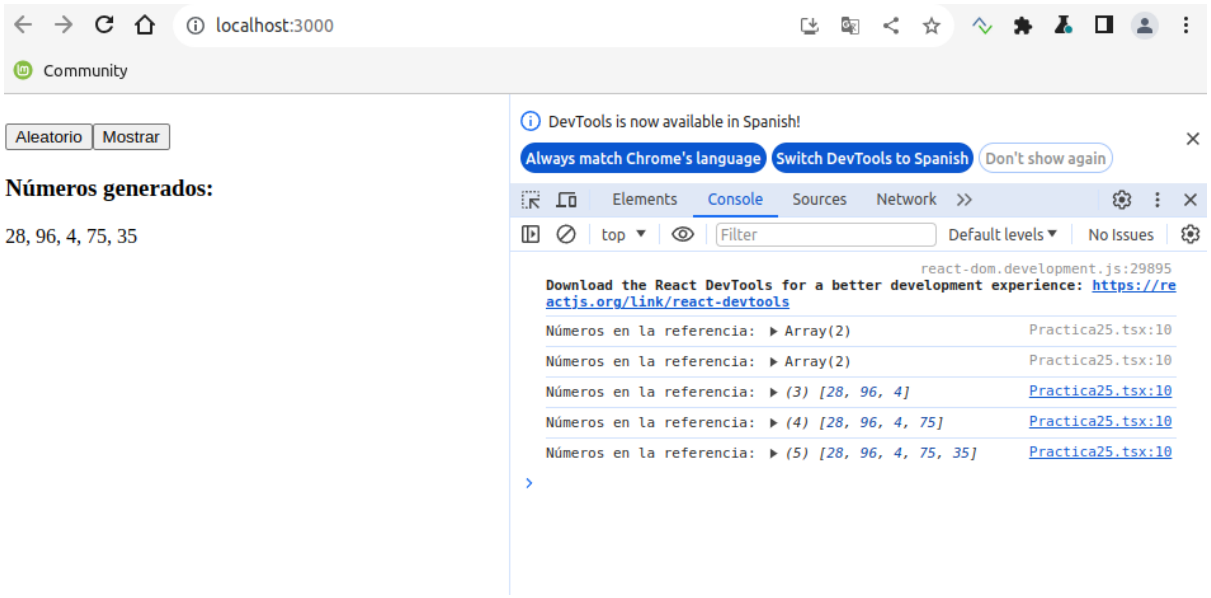
```
<h2>Juego de Acertar el Número Secreto</h2>
<p>{mensaje}</p>
<div>
<input
type="number"
ref={inputRef}
placeholder="Ingresa un número"
/>
<button onClick={apostar}>Apostar</button>
</div>
<button onClick={generarNuevoNumeroSecreto} style={{ marginTop: '10px' }}>
Reiniciar Juego
</button>
</div>
);
};
```

Práctica 25

Crear un functional component con dos botones uno dice: aleatorio que cada vez que se pulsa, agrega un aleatorio a un array apuntado por useRef() y otro botón que

dice: mostrar este último botón copia el array almacenado en la referencia y lo pone en el state. Mostrándose así el array de números generados

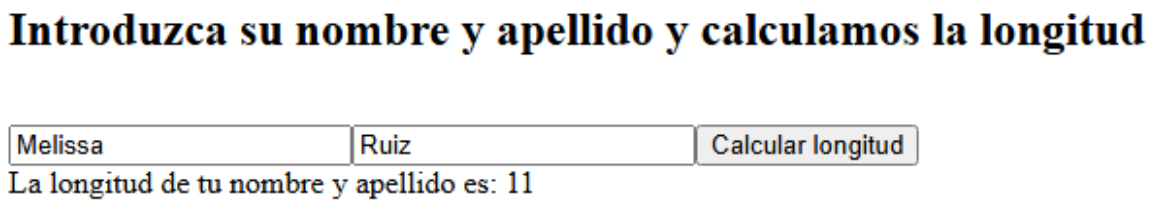
```
const Practica25 = () => {
const [numerosAleatorios, setNumerosAleatorios] = useState<number[]>([]);
const numerosRef = useRef<number[]>([]);

const agregarAleatorio = () => {
const numeroAleatorio = Math.floor(Math.random() * 100) + 1;
numerosRef.current.push(numeroAleatorio);
console.log('Números en la referencia:', numerosRef.current);
};

const mostrarNumeros = () => {
setNumerosAleatorios([...numerosRef.current]);
};

return (
<div>
<button onClick={agregarAleatorio}>Aleatorio</button>
<button onClick={mostrarNumeros}>Mostrar</button>
<div>
<h3>Números generados:</h3>
<p>{numerosAleatorios.join(', ')}</p>
</div>
</div>
);
};
```



Práctica 23

Usando useRef(), crear un componente con 2 input y un párrafo (etiqueta:

) donde uno de los inputs sea para el nombre y el otro input para los apellidos. Al pulsar en el botón tomará la información de los dos inputs y lo mostrará en el párrafo concatenados y dirá cuántas letras tiene el nombre completo



Práctica 24

Modificar el ejercicio de acertar número. Ahora en lugar de 10 botones, habrá un único input y un único botón. Al pulsar el botón en la acción que desencadene se usará useRef() para tomar la información que haya en el input y así realizar la apuesta

Modifico comentado para explicar

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

const Memorize = () => {
  // Estado para almacenar el número secreto
```

```
  const [numeroSecreto, setNumeroSecreto] = useState<number | null>(null);
  // Estado para almacenar el mensaje de resultado (si acertó, mayor o menor)
  const [mensaje, setMensaje] = useState<string>('');
  // useRef para manejar el valor del input
  const inputRef = useRef<HTMLInputElement>(null);

  // Generar número secreto cuando el componente se monta
  useEffect(() => {
    const numeroAleatorio = Math.floor(Math.random() * 10); // Número entre 0
y 9
    setNumeroSecreto(numeroAleatorio);
  }, []);

  // Función que se ejecuta al hacer la apuesta
  const apostar = () => {
    if (numeroSecreto !== null && inputRef.current) {
      const numeroApostado = parseInt(inputRef.current.value, 10);

      if (isNaN(numeroApostado)) {
        setMensaje('Por favor, introduce un número válido.');
        return;
      }

      if (numeroApostado === numeroSecreto) {
        setMensaje('¡Felicidades, acertaste!');
      } else if (numeroApostado < numeroSecreto) {
        setMensaje('El número secreto es mayor. ¡Inténtalo de nuevo!');
      } else {
        setMensaje('El número secreto es menor. ¡Inténtalo de nuevo!');
      }
    }
  };

  return (
    <div>
      <h2>Juego de Acertar el Número Secreto</h2>
      <p>{mensaje}</p>
      <div>
        {/* Input controlado mediante useRef */}
        <input ref={inputRef} type="number" placeholder="Introduce tu número"
/>
        <button onClick={apostar}>Apostar</button>
      </div>
    </div>
  );
};
```

```
export default Memorize;
```

Juego de Acertar el Número Secreto

El número secreto es mayor. ¡Inténtalo de nuevo!

3 Apostar

Práctica 25

Crear un functional component con dos botones uno dice: aleatorio que cada vez que se pulsa, agrega un aleatorio a un array apuntado por useRef() y otro botón que dice: mostrar este último botón copia el array almacenado en la referencia y lo pone en el state. Mostrándose así el array de números generados

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

const Practica25: React.FC = () => {
  // Estado para almacenar el array que se muestra
  const [numeros, setNumeros] = useState<number[]>([]);
  // useRef para almacenar el array de números generados aleatoriamente
  const numerosRef = useRef<number[]>([]);

  // Función para generar un número aleatorio y agregarlo al array apuntado
por useRef
  const agregarAleatorio = () => {
    const nuevoNumero = Math.floor(Math.random() * 100); // Número aleatorio
entre 0 y 99
    numerosRef.current.push(nuevoNumero); // Agregar al array en la referencia
    console.log('Números en useRef:', numerosRef.current); // Para depuración
  };

  // Función para copiar el array de useRef al estado y mostrarlo
  const mostrarArray = () => {
    setNumeros([...numerosRef.current]); // Actualizar el estado con los
valores de la referencia
  };

  return (
```

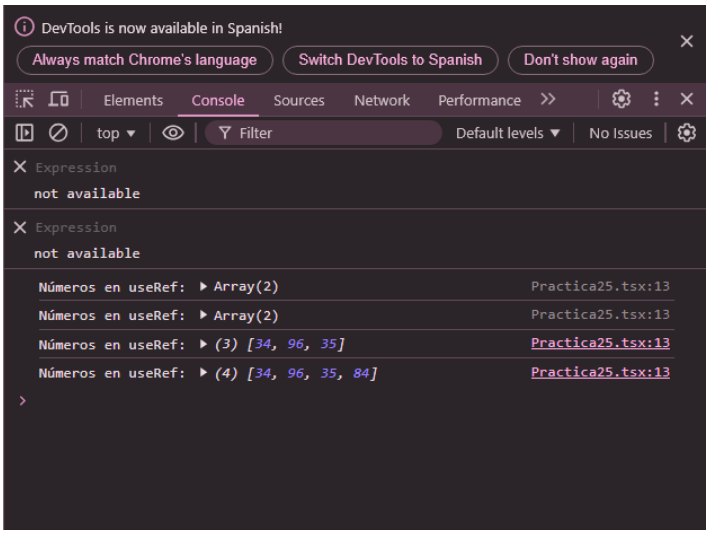
```
    <div>
      <h2>Práctica 25: Generar y Mostrar Números Aleatorios</h2>
      <button onClick={agregarAleatorio}>Aleatorio</button>
      <button onClick={mostrarArray}>Mostrar</button>
      <div>
        <h3>Números Generados:</h3>
        <ul>
          {numeros.map((numero, index) => (
            <li key={index}>{numero}</li>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default Practica25;
```

Práctica 25: Generar y Mostrar Números Aleatorios

Aleatorio | Mostrar

Números Generados:

- 34
- 96
- 35
- 84



Práctica 26

Crear un componente que tenga un cuadro de texto y un botón. Cuando se pulse en el botón se cargará otro componente debajo del botón que será la tabla de multiplicar (del 1 al 10) si es un número lo introducido. Si en lugar de un número fuera una palabra, entonces se cargará otro componente que nos dirá la cantidad de letras de la palabra y la cantidad de mayúsculas y minúsculas. Pasar la información a esos dos componentes mediante props

```
import React, { useState, useRef } from 'react';
import ContarLetras from './ContarLetras';
import TablaMultiplicar from './TablaMultiplicar';
```

```
type Props = {};

const Practica26 = ({}: Props) => {
  const inputRef = useRef<HTMLInputElement>(null); // Referencia para el input
  const [opcion, setOpcion] = useState<boolean>(true); // Controla qué
componente renderizar
  const [valor, setValor] = useState<string>(''); // Almacena el valor
introducido

  const manejarSubmit = () => {
    const input = inputRef.current?.value || ''; // Obtiene el valor del input
    setValor(input);

    // Determina si es un número o una palabra
    if (isNaN(parseInt(input))) {
      setOpcion(true); // Es una palabra
    } else {
      setOpcion(false); // Es un número
    }
  };

  return (
    <div>
      <h2>Práctica 26</h2>
      <div>
        <input type="text" placeholder="Introduce un número o una palabra"
ref={inputRef} />
        <button onClick={manejarSubmit}>Enviar</button>
      </div>
      <div>
        {opcion ? (
          <ContarLetras palabra={valor} />
        ) : (
          <TablaMultiplicar numero={parseInt(valor)} />
        )}
      </div>
    </div>
  );
};

export default Practica26;
```

Práctica 26

6 Enviar

Tabla de multiplicar del 6

- 6 x 1 = 6
- 6 x 2 = 12
- 6 x 3 = 18
- 6 x 4 = 24
- 6 x 5 = 30
- 6 x 6 = 36
- 6 x 7 = 42
- 6 x 8 = 48
- 6 x 9 = 54
- 6 x 10 = 60

Práctica 26

melissa Enviar

La palabra "melissa" tiene 7 letras

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

const Cronometro: React.FC = () => {
  const [segundos, setSegundos] = useState<number>(0); // Segundos
introducidos por el usuario
  const [tiempoRestante, setTiempoRestante] = useState<number | null>(null);
// Tiempo restante en la cuenta atrás
  const intervaloRef = useRef<ReturnType<typeof setInterval> | null>(null); //
Referencia para el intervalo

  // Función para iniciar el cronómetro
  const iniciarCronometro = () => {
```

```
    if (segundos > 0 && tiempoRestante === null) {
      const tiempoFinal = new Date().getTime() + segundos * 1000; // Calcular
el tiempo final

      intervaloRef.current = setInterval(() => {
        const ahora = new Date().getTime();
        const diferencia = Math.max(0, Math.ceil((tiempoFinal - ahora) /
1000)); // Calcular tiempo restante

        setTiempoRestante(diferencia);

        if (diferencia <= 0) {
          detenerCronometro(); // Detener cuando llegue a 0
        }
      }, 1000);
    }
  };

  // Función para detener el cronómetro
  const detenerCronometro = () => {
    if (intervaloRef.current) {
      clearInterval(intervaloRef.current);
      intervaloRef.current = null;
    }
    setTiempoRestante(null);
  };

  return (
    <div>
      <h2>Cronómetro</h2>
      <div>
        <label htmlFor="segundos">Segundos:</label>
        <input
          id="segundos"
          type="number"
          value={segundos}
          onChange={(e) => setSegundos(Number(e.target.value))}
        />
      </div>
      <div>
        <button onClick={iniciarCronometro} disabled={tiempoRestante !==
null}>
          Iniciar
        </button>
        <button onClick={detenerCronometro} disabled={tiempoRestante ===
null}>
          Detener
```

```
        </button>
      </div>
      <div>
        {tiempoRestante === null ? (
          <p>Antes de empezar</p>
        ) : tiempoRestante > 0 ? (
          <p>Tiempo restante: {tiempoRestante} segundos</p>
        ) : (
          <p>¡Finalizado!</p>
        )}
      </div>
    </div>
  );
};

export default Cronometro;
```

Explicación del Código

1. **Estado y Referencias:**
   a. **`segundos`: Almacena el número de segundos introducido por el usuario.**
   b. **`tiempoRestante`: Almacena el tiempo restante durante la cuenta atrás.**
   c. **`intervaloRef`: Se usa para almacenar el identificador del intervalo creado por `setInterval` y detenerlo posteriormente.**
2. **Lógica del Cronómetro:**
   a. **Cuando el usuario hace clic en "Iniciar", se calcula el tiempo final sumando los segundos introducidos al tiempo actual (`new Date().getTime()`).**
   b. **Se actualiza el tiempo restante cada segundo con `setInterval`.**
   c. **Cuando el tiempo restante llega a 0, el cronómetro se detiene automáticamente.**
3. **Botones:**
   a. **"Iniciar" está deshabilitado mientras el cronómetro está en funcionamiento.**
   b. **"Detener" está deshabilitado si no hay un cronómetro activo.**
4. **Renderizado Dinámico:**
   a. **Muestra diferentes mensajes dependiendo del estado:**
      i. **"Antes de empezar" si el cronómetro no se ha iniciado.**
      ii. **"Tiempo restante" mientras está en marcha.**
      iii. **"¡Finalizado!" cuando la cuenta atrás termina.**

Cronómetro

Segundos: 600

Iniciar Detener

Tiempo restante: 596 segundos

Práctica 28

Realizar un componente llamado: MostrarInput que se introduzca el texto en el input y se vaya mostrando en un h5