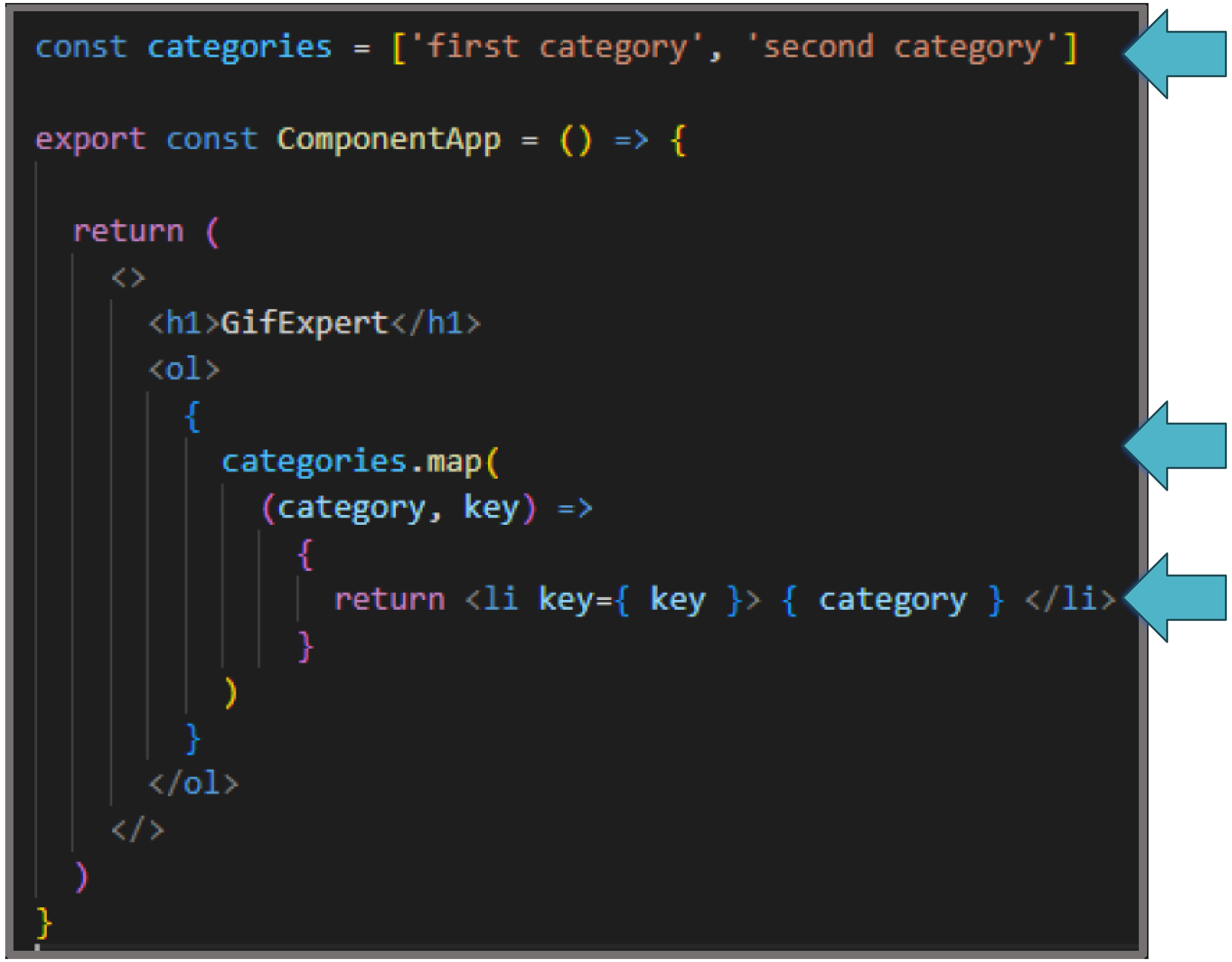
Estructura de Datos | |

Jonathan López Londoño
jlopezl@uao.edu.co
315 926 5443

React JS



```
const categories = ['first category', 'second category']

export const ComponentApp = () => {

  return (
    <>
      <h1>GifExpert</h1>
      <ol>
        {
          categories.map(
            (category, key) =>
              {
                return <li key={ key }> { category } </li>
              }
          )
        }
      </ol>
    </>
  )
}
```

Print Arrays into HTML

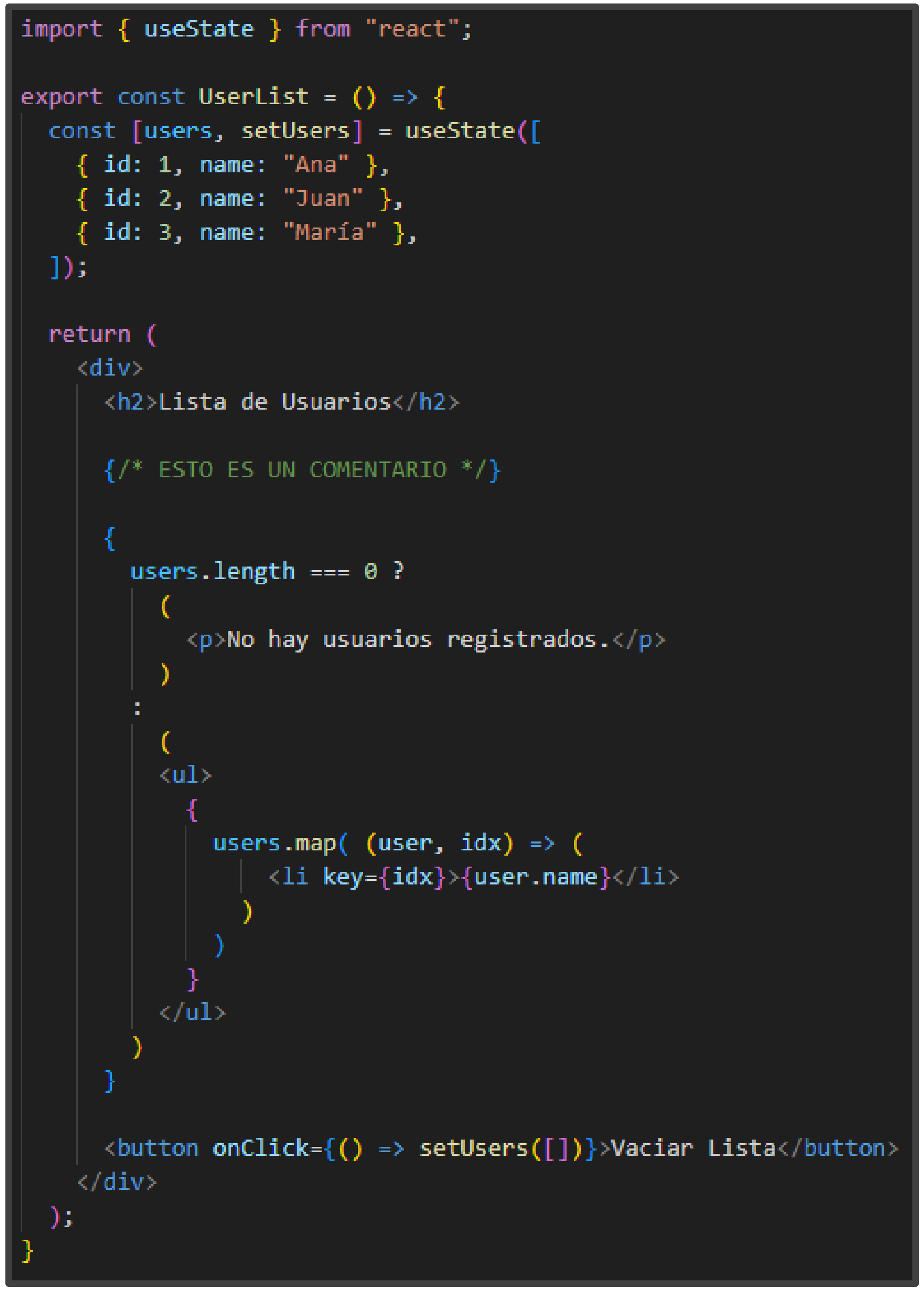
Conditional Rendering

You can use Null Checks to render conditionally.

Note:

You can add comments in rendering by using:

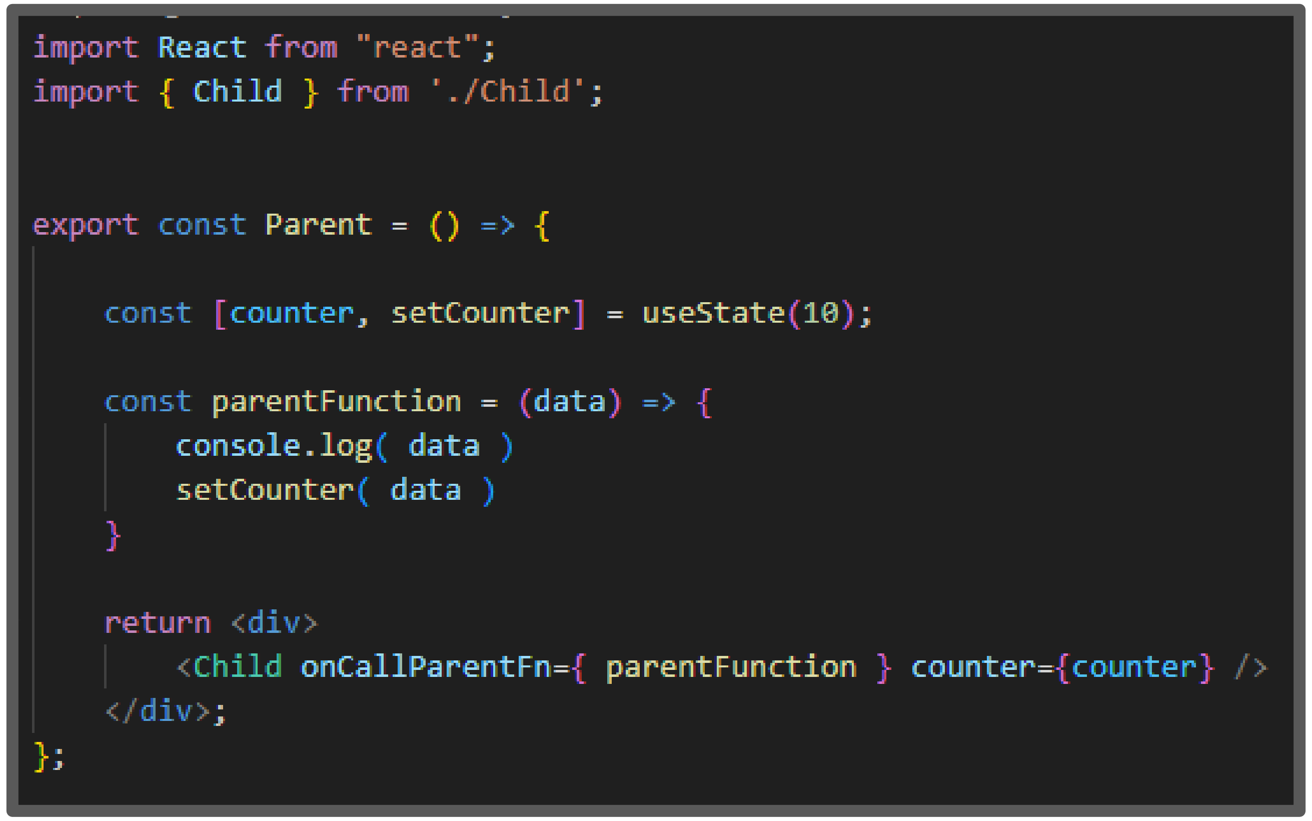
{/ COMMENT */}*



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

export const UserList = () => {
  const [users, setUsers] = useState([
    { id: 1, name: "Ana" },
    { id: 2, name: "Juan" },
    { id: 3, name: "María" },
  ]);

  return (
    <div>
      <h2>Lista de Usuarios</h2>

      {/* ESTO ES UN COMENTARIO */}

      {
        users.length === 0 ?
          (
            <p>No hay usuarios registrados.</p>
          )
        :
          (
          <ul>
            {
              users.map( (user, idx) => (
                  <li key={idx}>{user.name}</li>
                )
              )
            }
          </ul>
        )
      }

      <button onClick={() => setUsers([])}>Vaciar Lista</button>
    </div>
  );
}
```

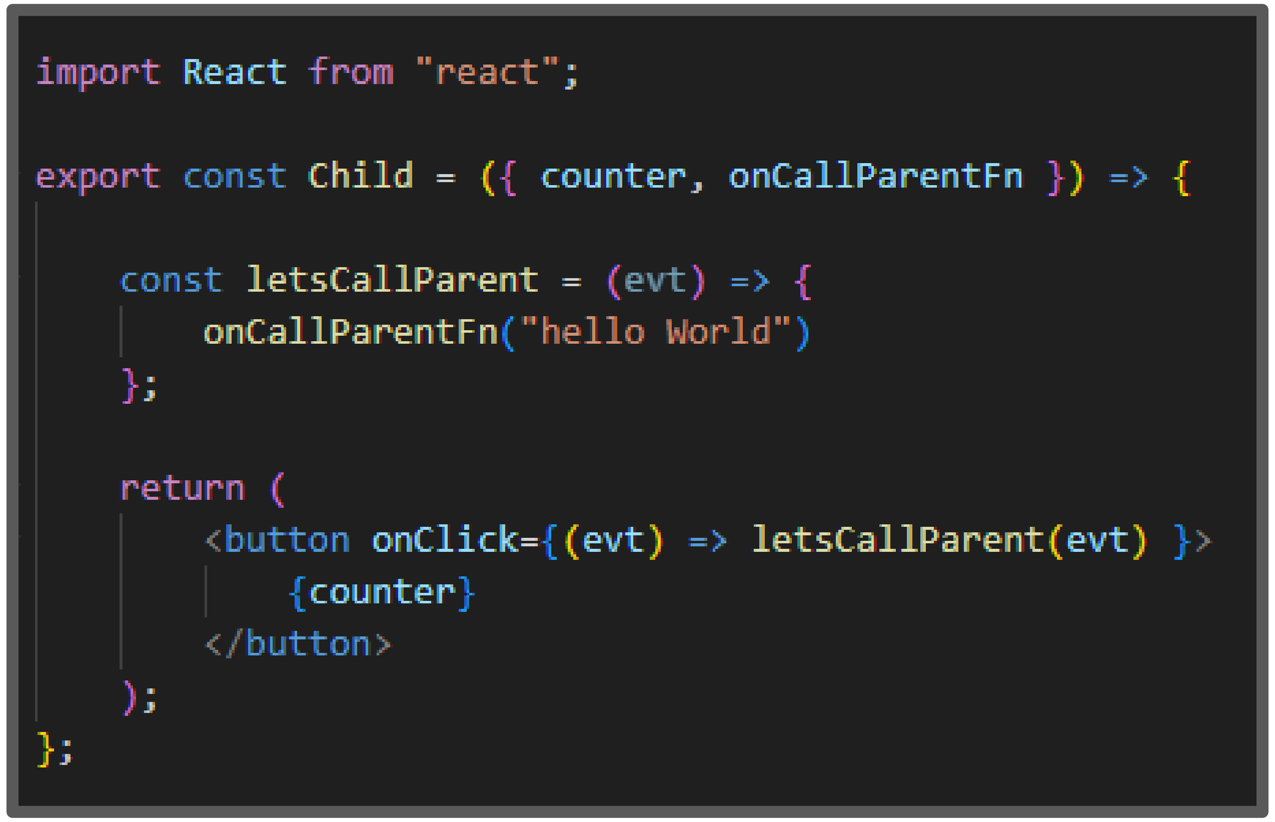
Communication Child to Parent

/Parent.jsx



```jsx
import React from "react";
import { Child } from './Child';


export const Parent = () => {

    const [counter, setCounter] = useState(10);

    const parentFunction = (data) => {
        console.log( data )
        setCounter( data )
    }

    return <div>
        <Child onCallParentFn={ parentFunction } counter={counter} />
    </div>;
};
```

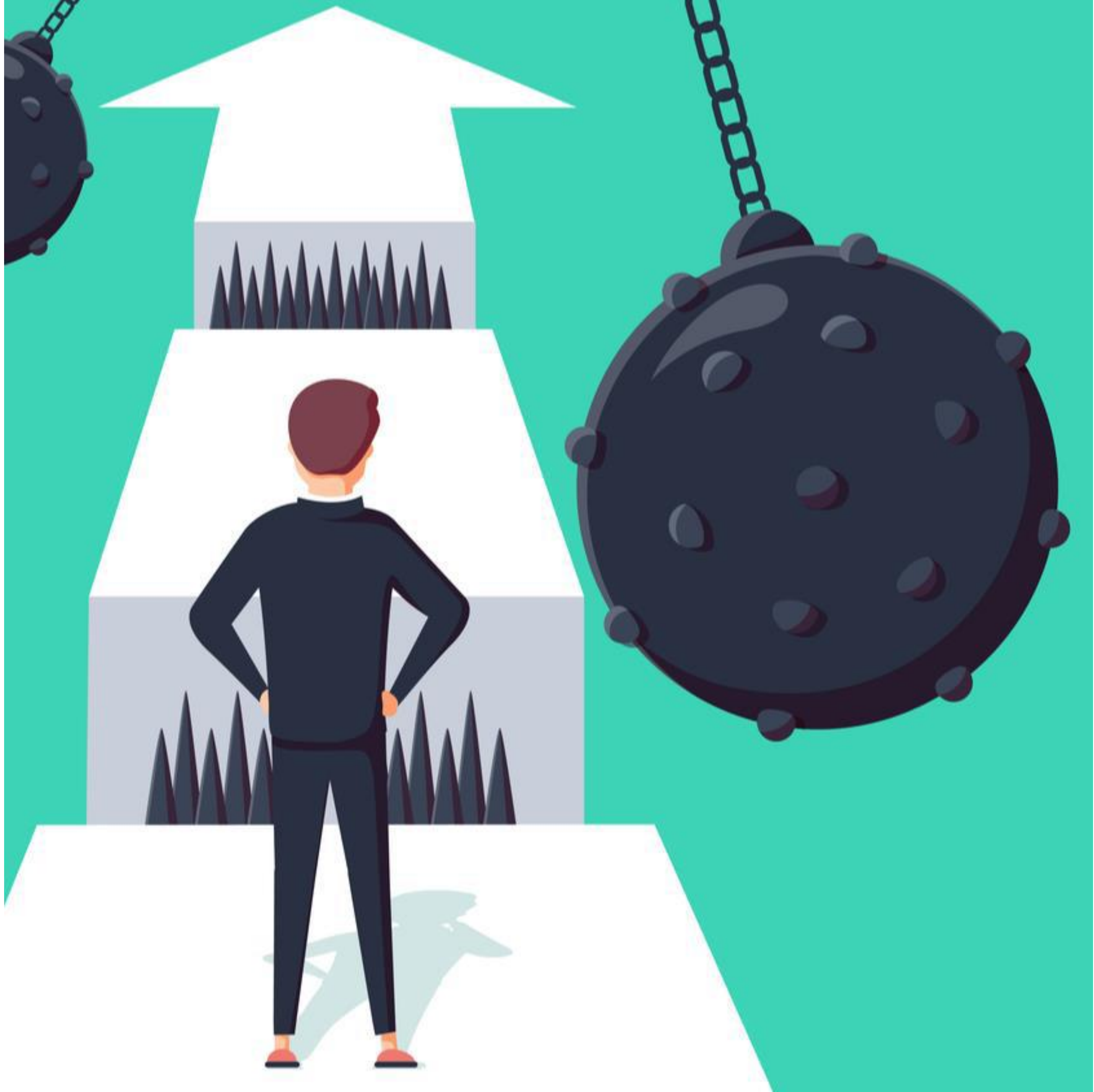
/Child.jsx



```jsx
import React from "react";

export const Child = ({ counter, onCallParentFn }) => {

    const letsCallParent = (evt) => {
        onCallParentFn("hello World")
    };

    return (
        <button onClick={(evt) => letsCallParent(evt) }>
            {counter}
        </button>
    );
};
```

CHALLENGE 04

1. Use ComponentApp. Add a new input tag to write some text, use onChange event to get changes on input
2. Handle above input with useState hook to setCategory
3. Send event variable to setCategory function to get value from input
4. Add a new Button to add categories.
5. Use useState hook and call setCategories to add the current category to the list by using onClick event button.
6. Inside above function, call setCategory to clear input field after add one category
7. Divide the component in Parent and Child

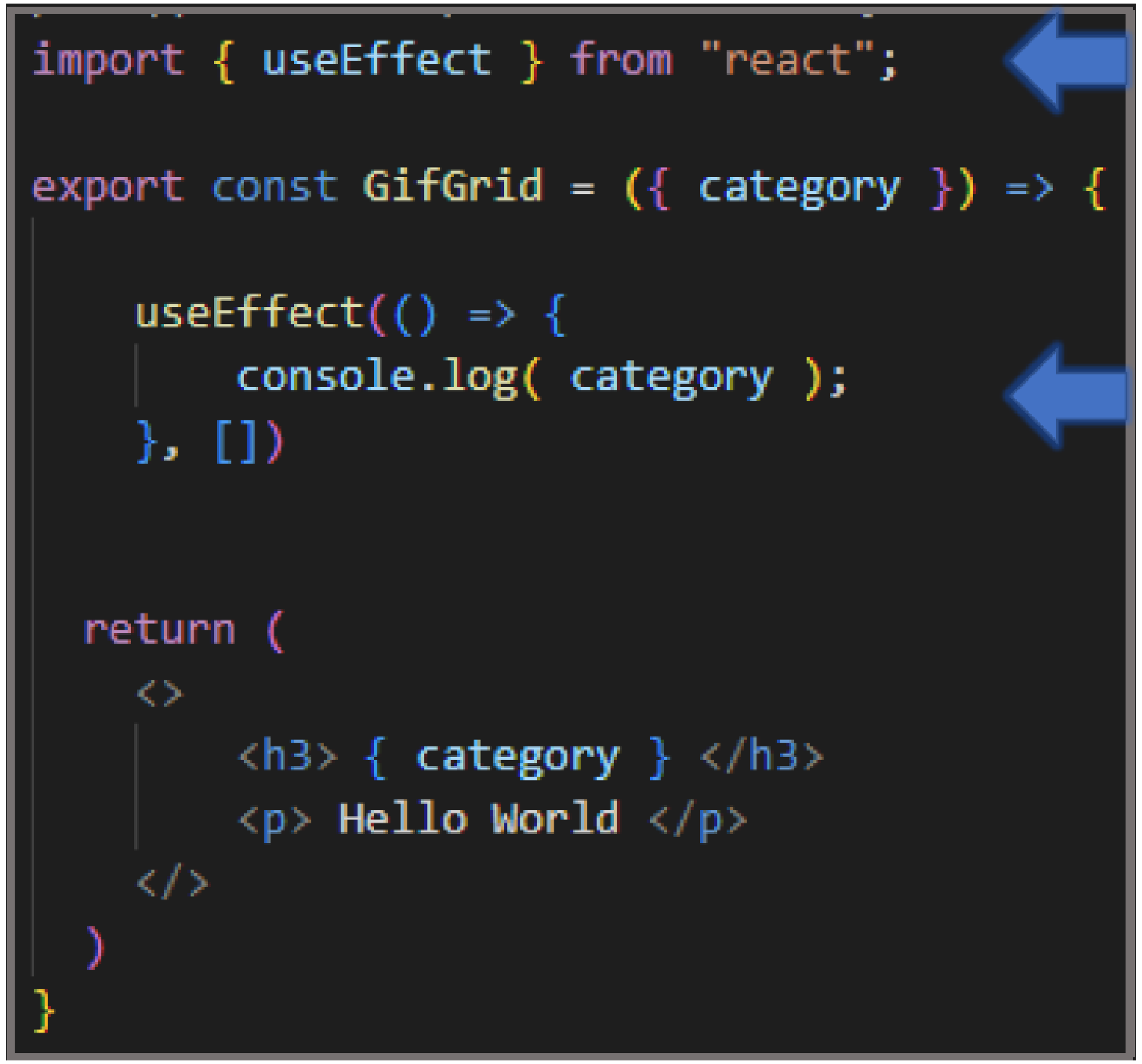


Hooks

UseEffect: When using this hook, React must execute something after rendering/updating according to list of variables to Watch

import {useEffect } from 'react'

useEffect(() => {
// TODO
},[...VARIABLES TO WATCH])



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

export const GifGrid = ({ category }) => {

    useEffect(() => {
        console.log( category );
    }, [])


  return (
    <>
        <h3> { category } </h3>
        <p> Hello World </p>
    </>
  )
}
```

Components Life Cycle

UseEffect *Examples:*

```
useEffect( () => {
    console.log("El componente se ha montado.")
},[])
```

Will be executed only once, when the component is mounted

```
useEffect( () => {
    return () => {
        console.log("Componente desmontado.");
    }

},[])
```

Will be executed only once the component is unmounted

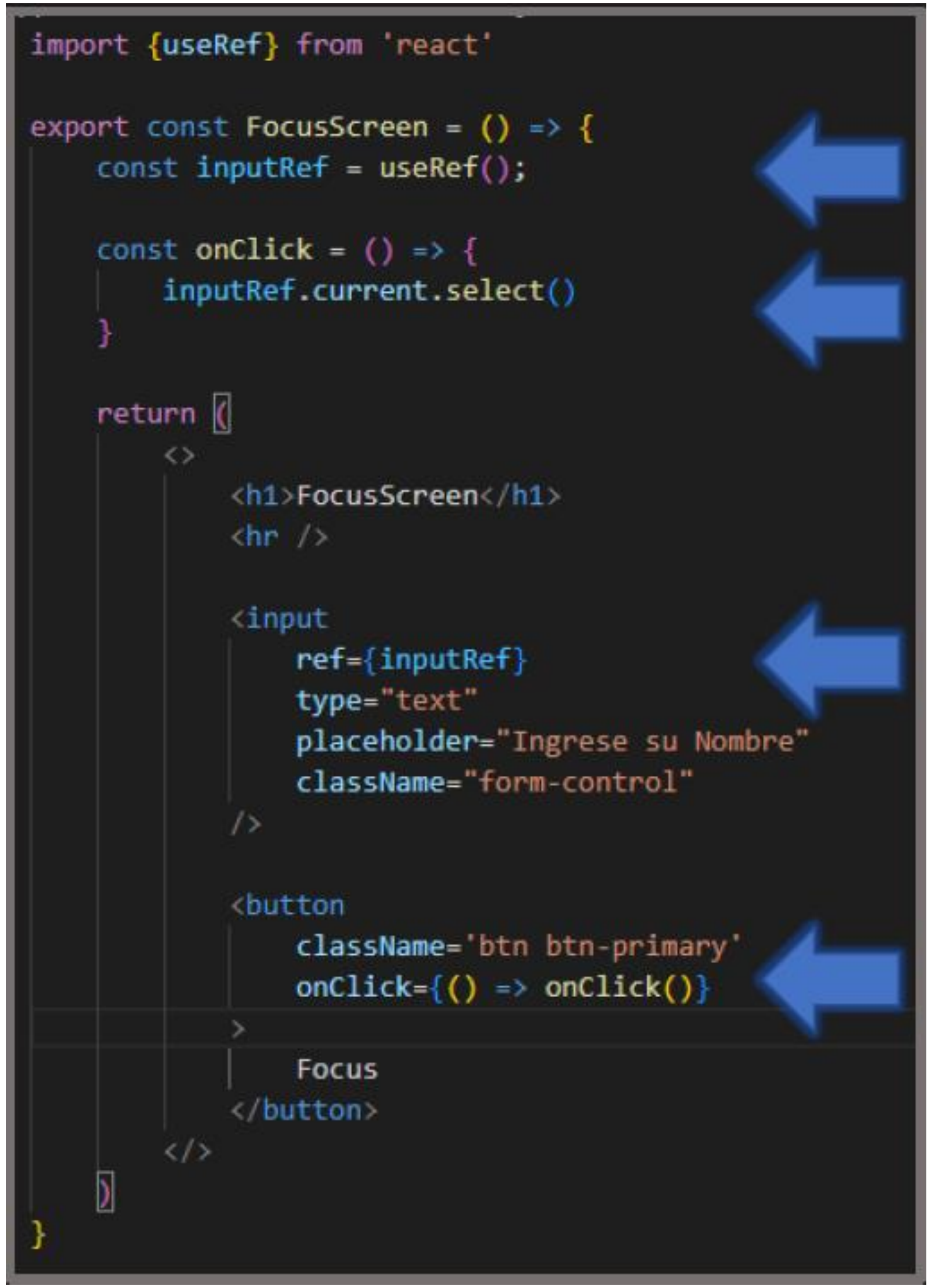
```
useEffect( () => {
    console.log("Contador actualizado: ${counter}"
)},[counter])
```

Will be executed when the counter's value has changed

Hooks

UseRef returns a mutable ref object whose *.current* property is initialized with the passed argument (initialValue).

The returned object will remain persistent for the entire life of the component.

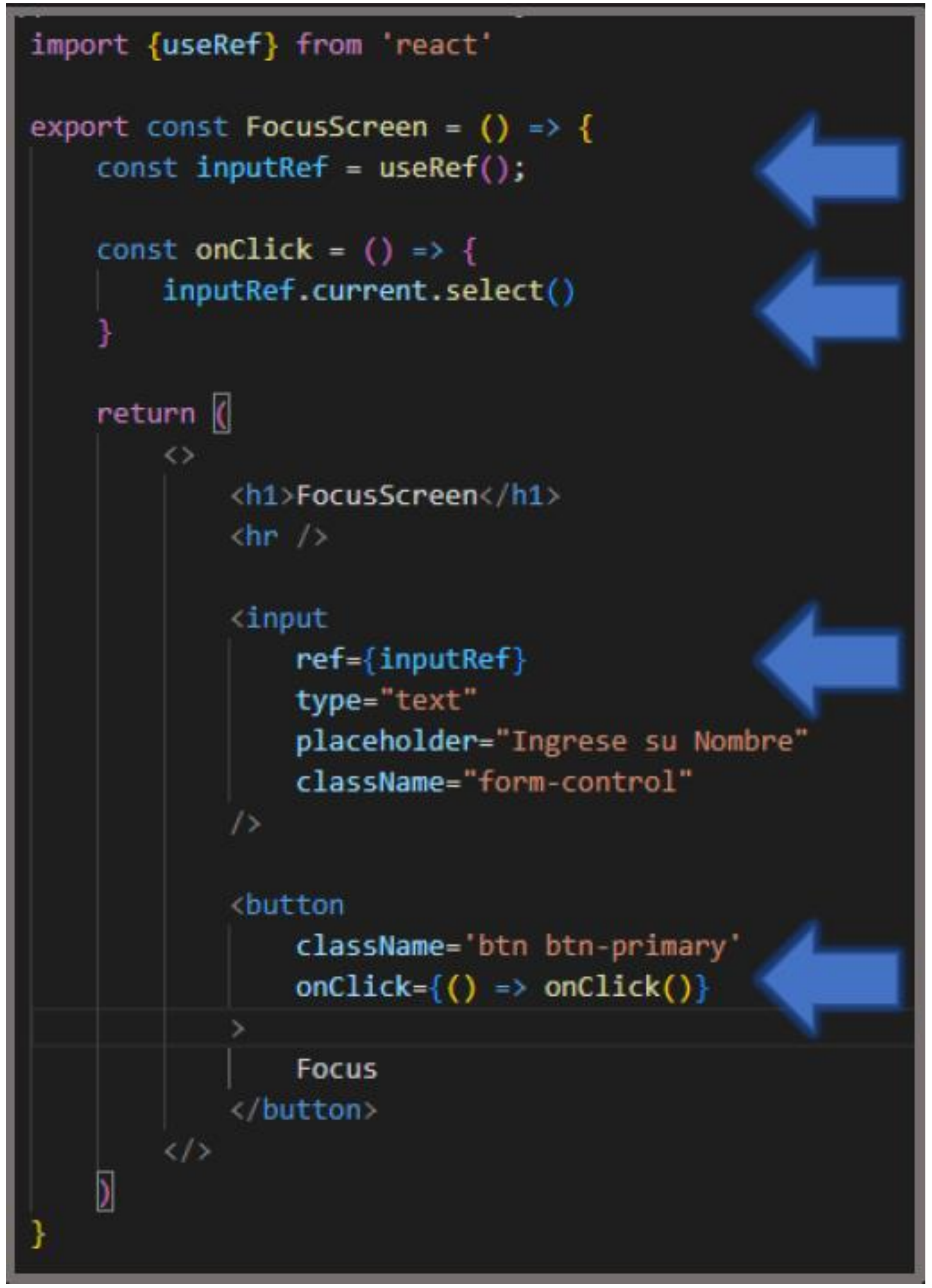


```
import {useRef} from 'react'

export const FocusScreen = () => {
    const inputRef = useRef();

    const onClick = () => {
        inputRef.current.select()
    }

    return (
        <>
            <h1>FocusScreen</h1>
            <hr />

            <input
                ref={inputRef}
                type="text"
                placeholder="Ingrese su Nombre"
                className="form-control"
            />

            <button
                className='btn btn-primary'
                onClick={() => onClick()}
            >
                Focus
            </button>
        </>
    )
}
```

Hooks

UseRef returns a mutable ref object whose *.current* property is initialized with the passed argument (initialValue).

The returned object will remain persistent for the entire life of the component.

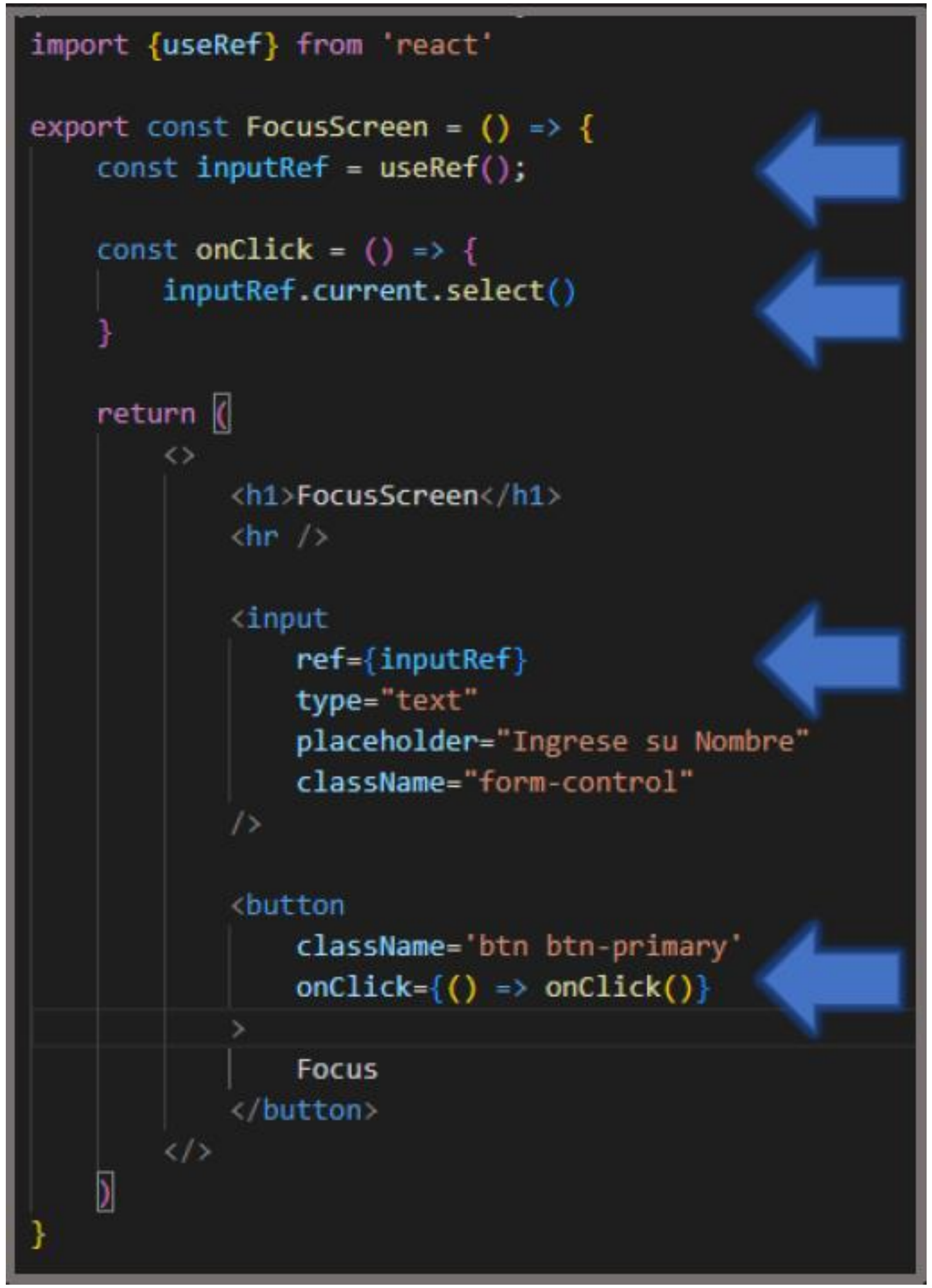


```
import {useRef} from 'react'

export const FocusScreen = () => {
    const inputRef = useRef();

    const onClick = () => {
        inputRef.current.select()
    }

    return (
        <>
            <h1>FocusScreen</h1>
            <hr />

            <input
                ref={inputRef}
                type="text"
                placeholder="Ingrese su Nombre"
                className="form-control"
            />

            <button
                className='btn btn-primary'
                onClick={() => onClick()}
            >
                Focus
            </button>
        </>
    )
}
```

Hooks

UseRef returns a mutable ref object whose *.current* property is initialized with the passed argument (initialValue).

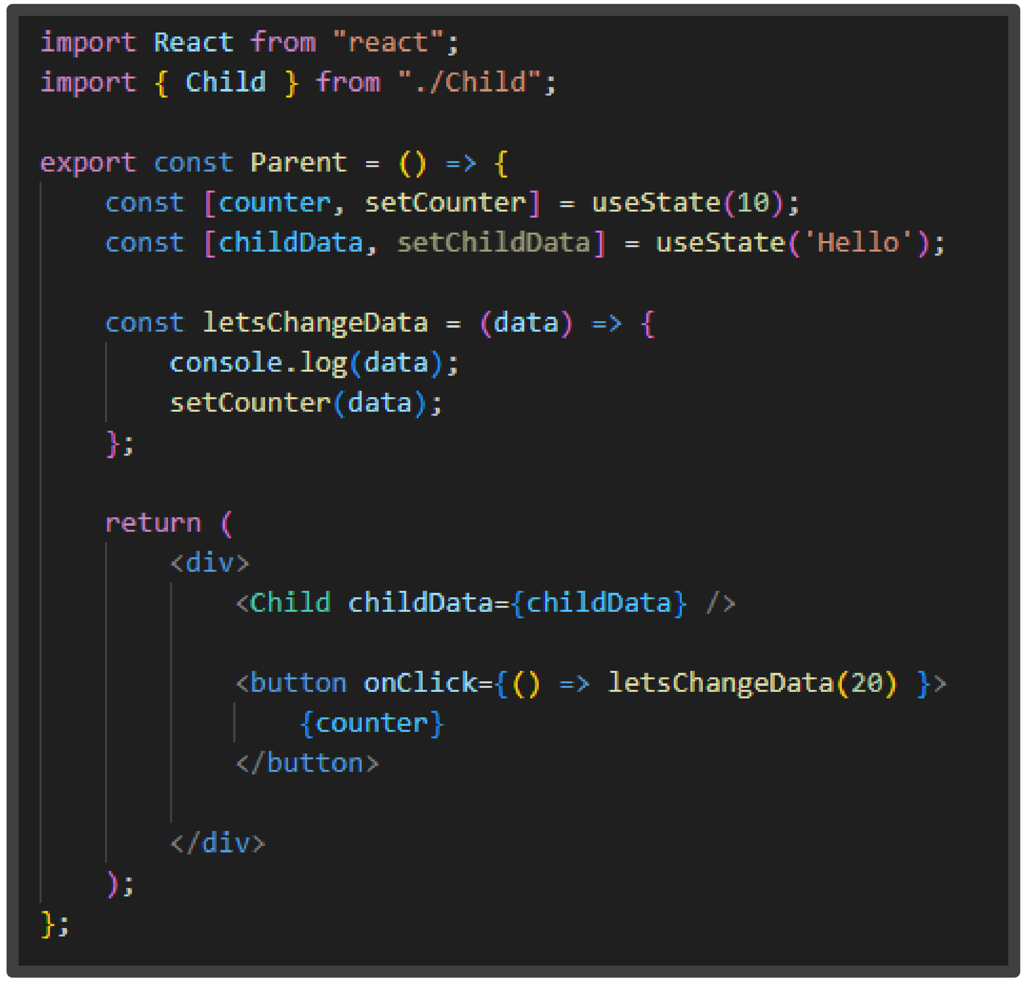
The returned object will remain persistent for the entire life of the component.



```
import {useRef} from 'react'

export const FocusScreen = () => {
    const inputRef = useRef();

    const onClick = () => {
        inputRef.current.select()
    }

    return (
        <>
            <h1>FocusScreen</h1>
            <hr />

            <input
                ref={inputRef}
                type="text"
                placeholder="Ingrese su Nombre"
                className="form-control"
            />

            <button
                className='btn btn-primary'
                onClick={() => onClick()}
            >
                Focus
            </button>
        </>
    )
}
```

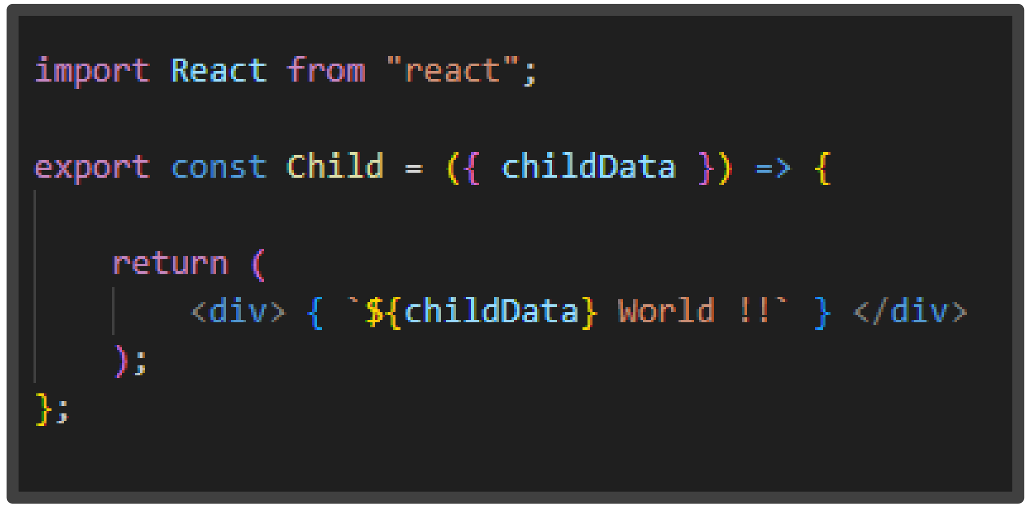
Communication Child to Parent

/Parent.jsx



```jsx
import React from "react";
import { Child } from "./Child";

export const Parent = () => {
    const [counter, setCounter] = useState(10);
    const [childData, setChildData] = useState('Hello');

    const letsChangeData = (data) => {
        console.log(data);
        setCounter(data);
    };

    return (
        <div>
            <Child childData={childData} />

            <button onClick={() => letsChangeData(20) }>
                {counter}
            </button>

        </div>
    );
};
```

/Child.jsx



```jsx
import React from "react";

export const Child = ({ childData }) => {

    return (
        <div> { `${childData} World !!` } </div>
    );
};
```

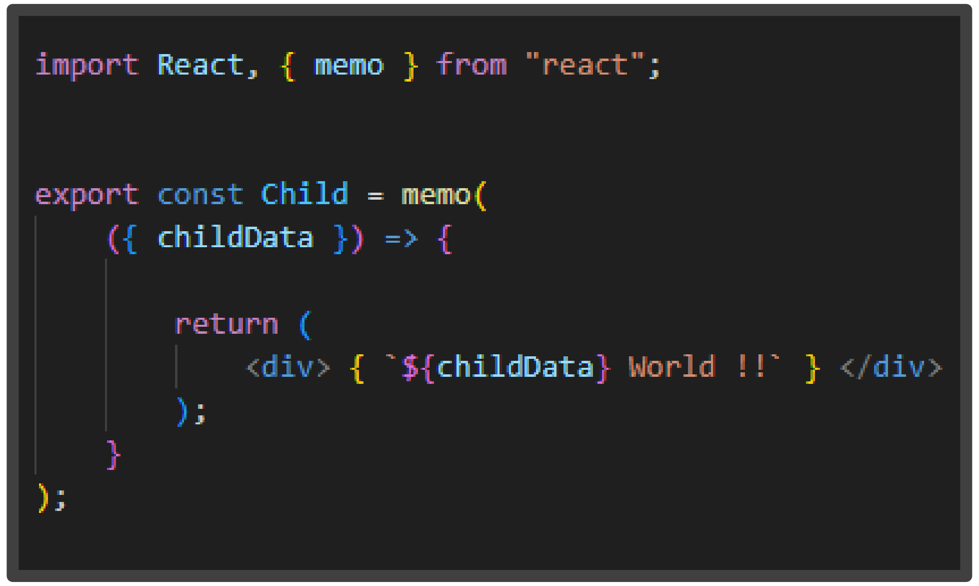
When parent is reloaded, all Children are reloaded, even if their props hasn't changed.

Hooks

Memo: Use it when you don’t need to render child components after change any state from parent component.

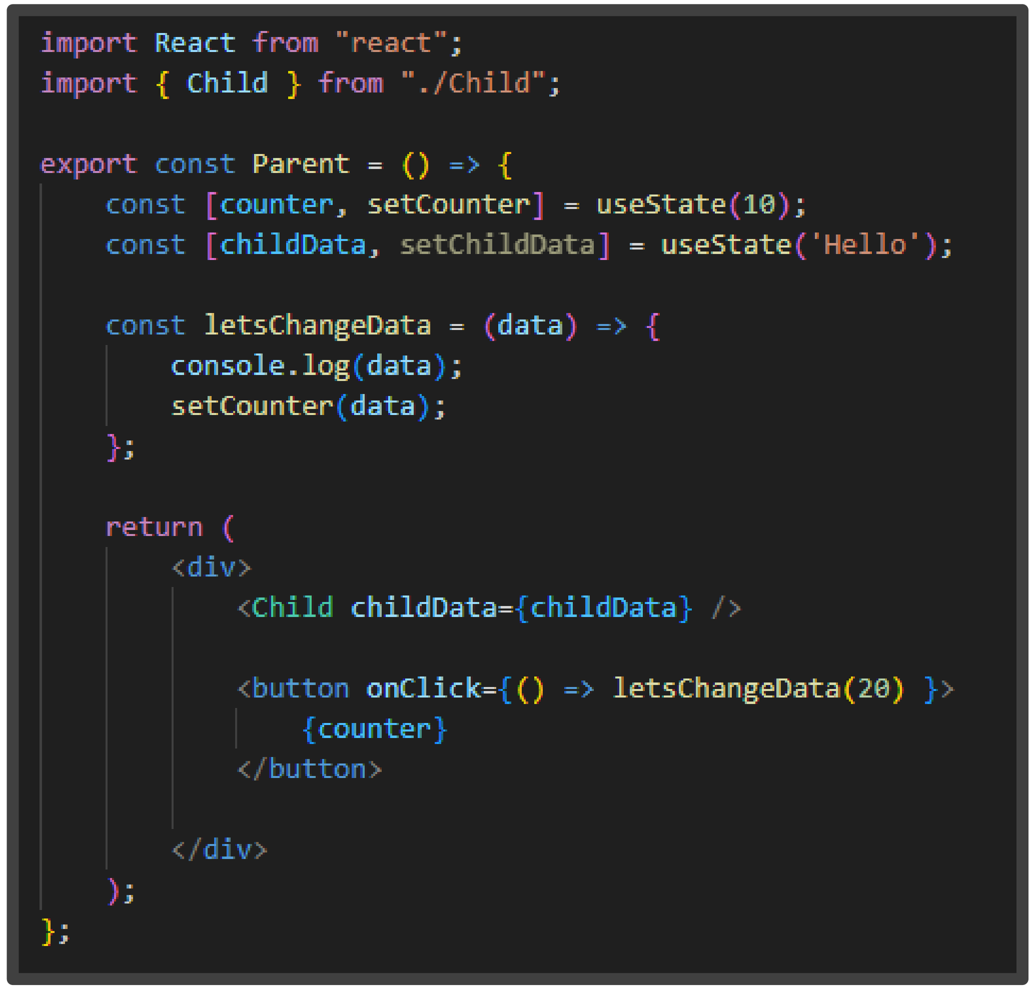
Child components will be rendered only when its props change.

/Child.jsx



```jsx
import React, { memo } from "react";


export const Child = memo(
    ({ childData }) => {

        return (
            <div> { `${childData} World !!` } </div>
        );
    }
);
```

/Parent.jsx



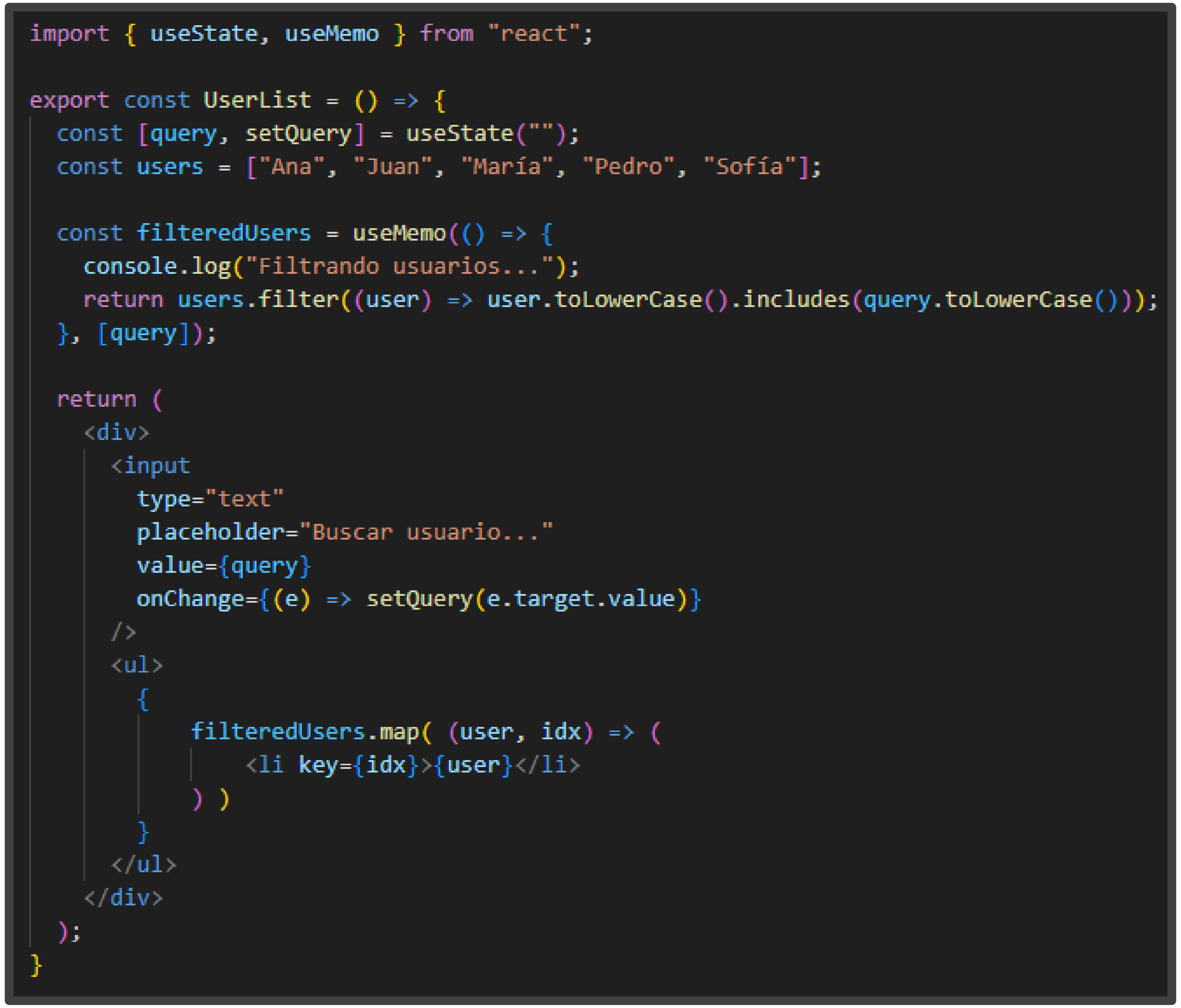
```jsx
import React from "react";
import { Child } from "./Child";

export const Parent = () => {
    const [counter, setCounter] = useState(10);
    const [childData, setChildData] = useState('Hello');

    const letsChangeData = (data) => {
        console.log(data);
        setCounter(data);
    };

    return (
        <div>
            <Child childData={childData} />

            <button onClick={() => letsChangeData(20) }>
                {counter}
            </button>

        </div>
    );
};
```

Hooks

UseMemo: This hook will memorize the result of a heavy function and will call it again only when one of its dependencies has changed and the new dependence value hasn't been calculated.

Without useMemo, every render would recalculate filteredUsers, even if query doesn't change.

With useMemo, it is only recalculated when query changes

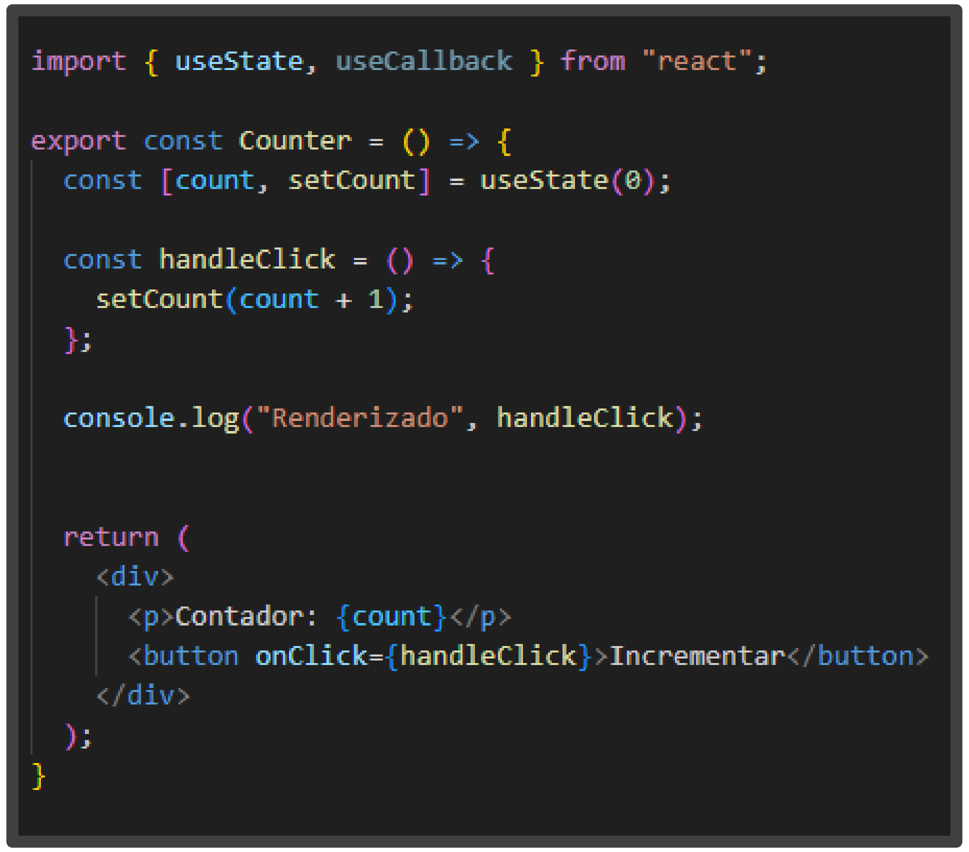


```
import { useState, useMemo } from "react";

export const UserList = () => {
  const [query, setQuery] = useState("");
  const users = ["Ana", "Juan", "María", "Pedro", "Sofía"];

  const filteredUsers = useMemo(() => {
    console.log("Filtrando usuarios...");
    return users.filter((user) => user.toLowerCase().includes(query.toLowerCase()));
  }, [query]);

  return (
    <div>
      <input
        type="text"
        placeholder="Buscar usuario..."
        value={query}
        onChange={(e) => setQuery(e.target.value)}
      />
      <ul>
        {
            filteredUsers.map( (user, idx) => (
              <li key={idx}>{user}</li>
            ) )
        }
      </ul>
    </div>
  );
}
```

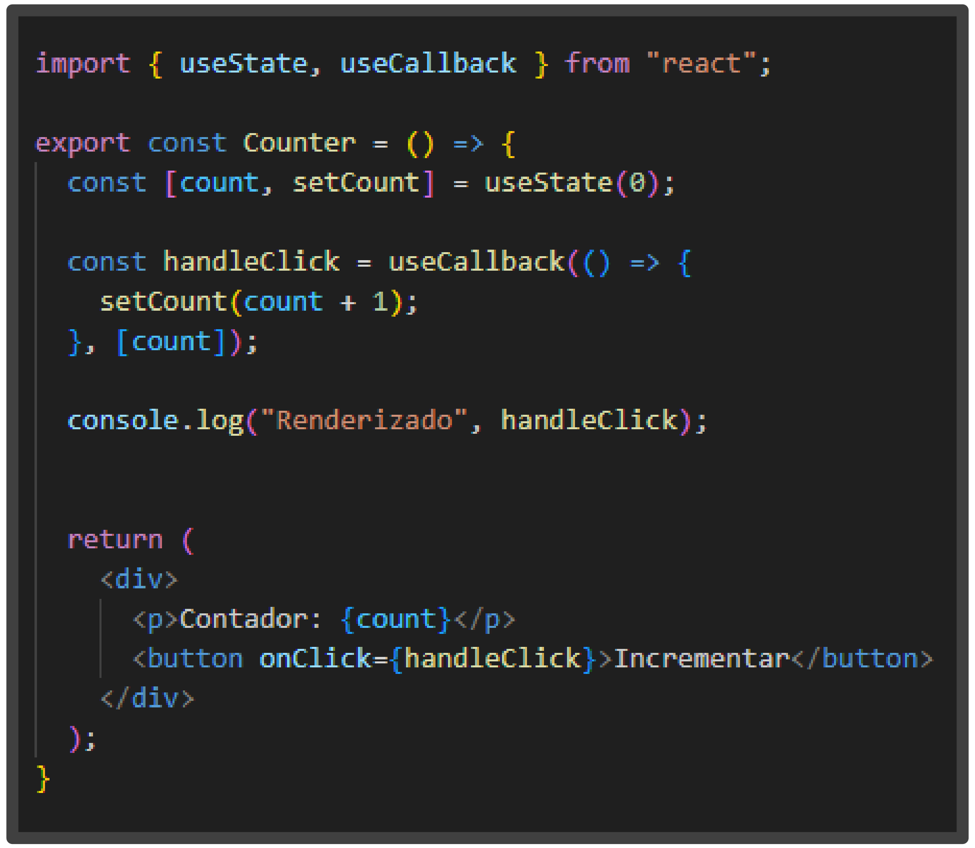
Hooks

UseCallback: This hook will memorize a version of the callback that only changes if one of the dependencies has changed.

This is useful when passing callbacks to optimized child components that depends on reference equality to avoid unnecessary renders.



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

export const Counter = () => {
  const [count, setCount] = useState(0);

  const handleClick = () => {
    setCount(count + 1);
  };

  console.log("Renderizado", handleClick);


  return (
    <div>
      <p>Contador: {count}</p>
      <button onClick={handleClick}>Incrementar</button>
    </div>
  );
}
```

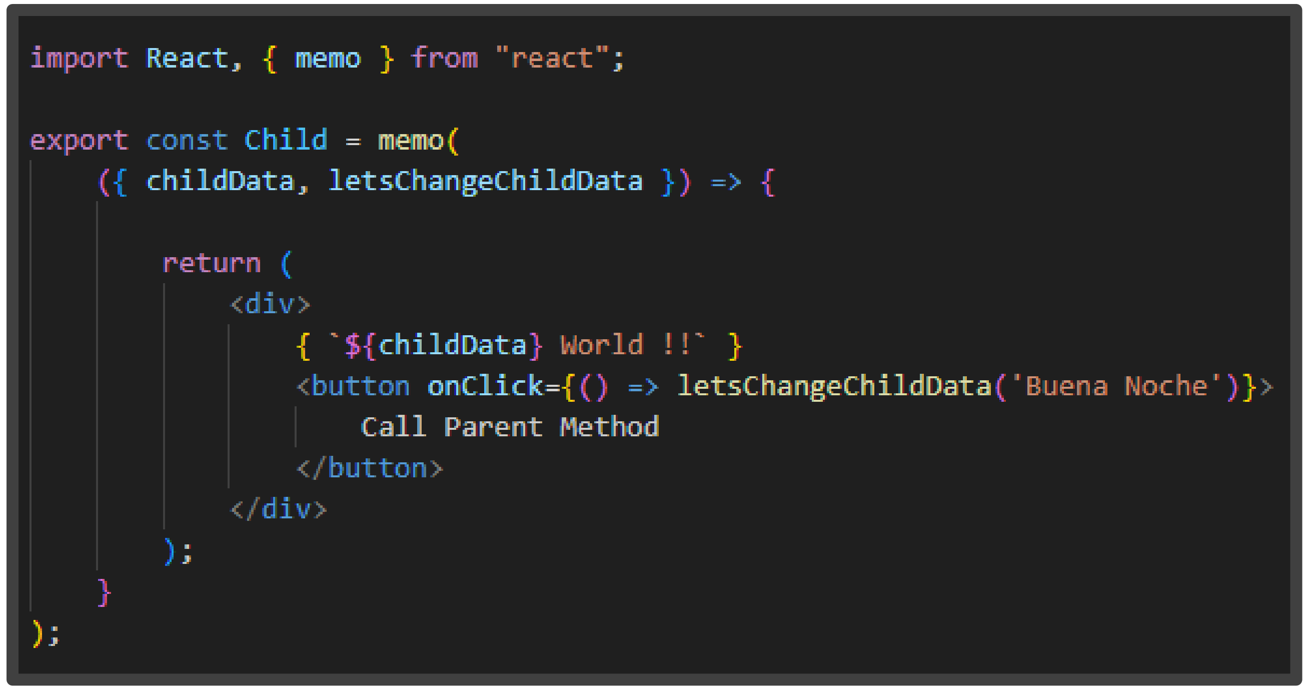


```
import { useState, useCallback } from "react";

export const Counter = () => {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    setCount(count + 1);
  }, [count]);

  console.log("Renderizado", handleClick);


  return (
    <div>
      <p>Contador: {count}</p>
      <button onClick={handleClick}>Incrementar</button>
    </div>
  );
}
```

Hooks

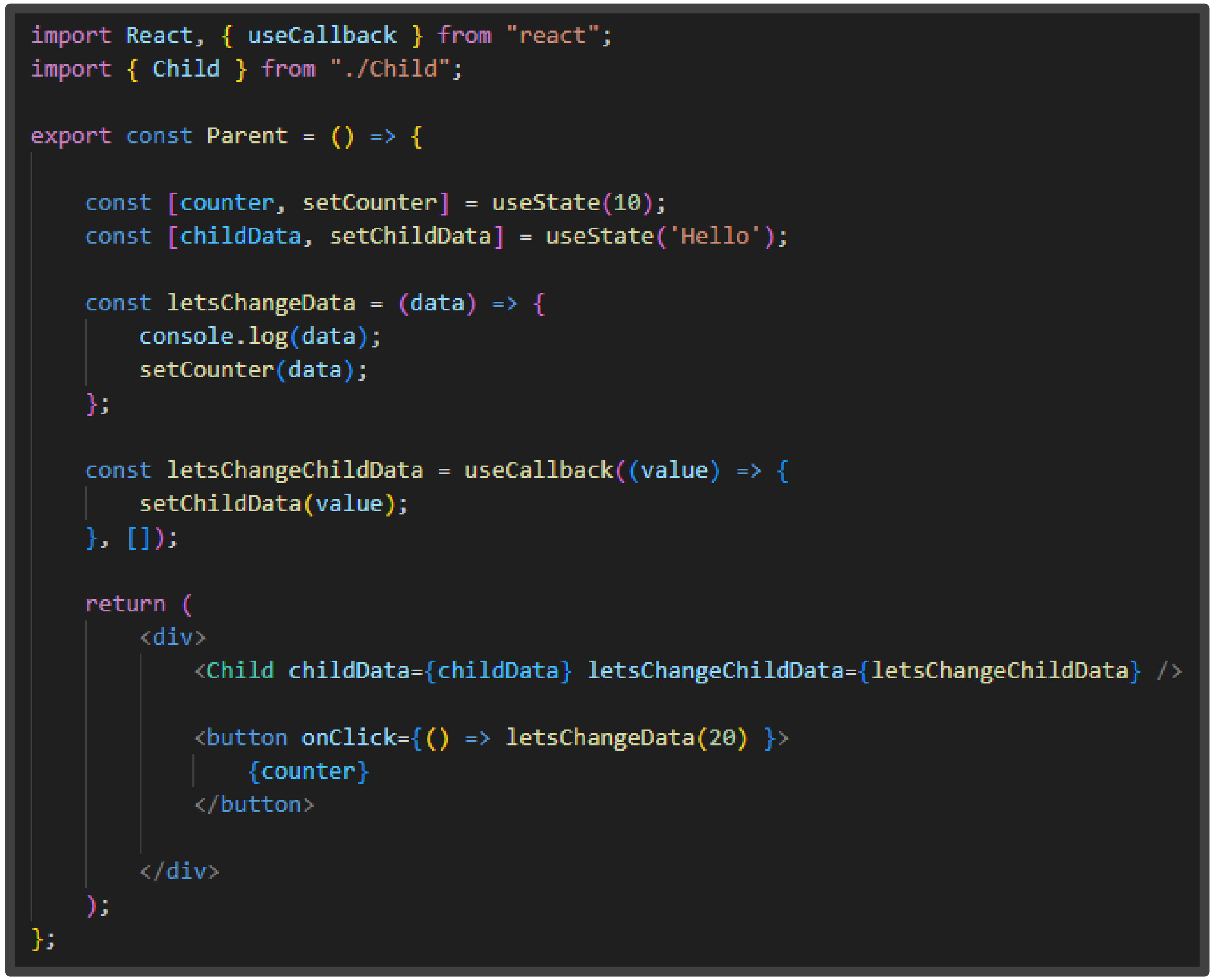
UseCallback: This is useful when passing callbacks to optimized child components that depends on reference equality to avoid unnecessary renders.

/Child.jsx

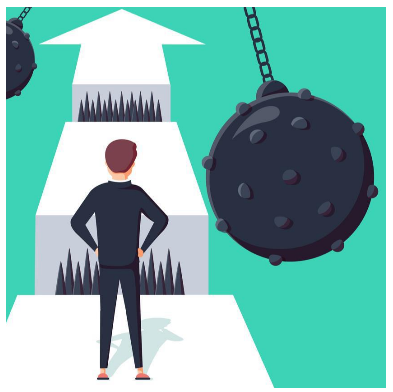


```jsx
import React, { memo } from "react";

export const Child = memo(
    ({ childData, letsChangeChildData }) => {

        return (
            <div>
                { `${childData} World !!` }
                <button onClick={() => letsChangeChildData('Buena Noche')}>
                    Call Parent Method
                </button>
            </div>
        );
    }
);
```

/Parent.jsx

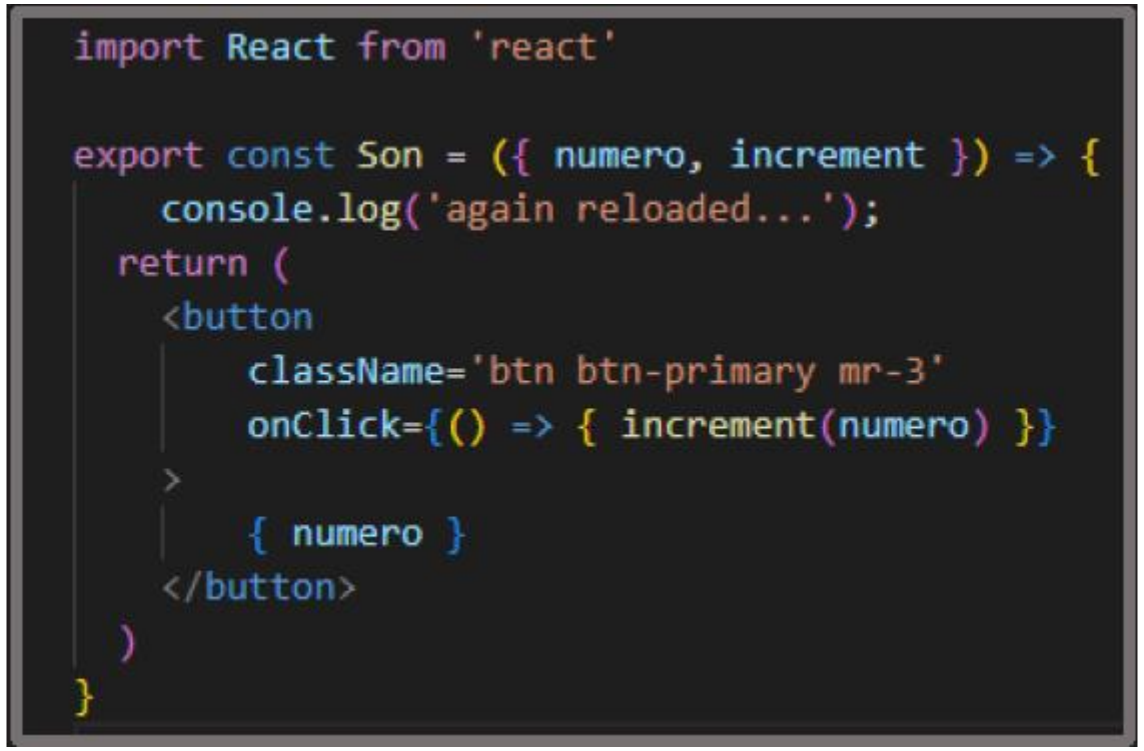


```jsx
import React, { useCallback } from "react";
import { Child } from "./Child";

export const Parent = () => {

    const [counter, setCounter] = useState(10);
    const [childData, setChildData] = useState('Hello');

    const letsChangeData = (data) => {
        console.log(data);
        setCounter(data);
    };

    const letsChangeChildData = useCallback((value) => {
        setChildData(value);
    }, []);

    return (
        <div>
            <Child childData={childData} letsChangeChildData={letsChangeChildData} />

            <button onClick={() => letsChangeData(20) }>
                {counter}
            </button>

        </div>
    );
};
```

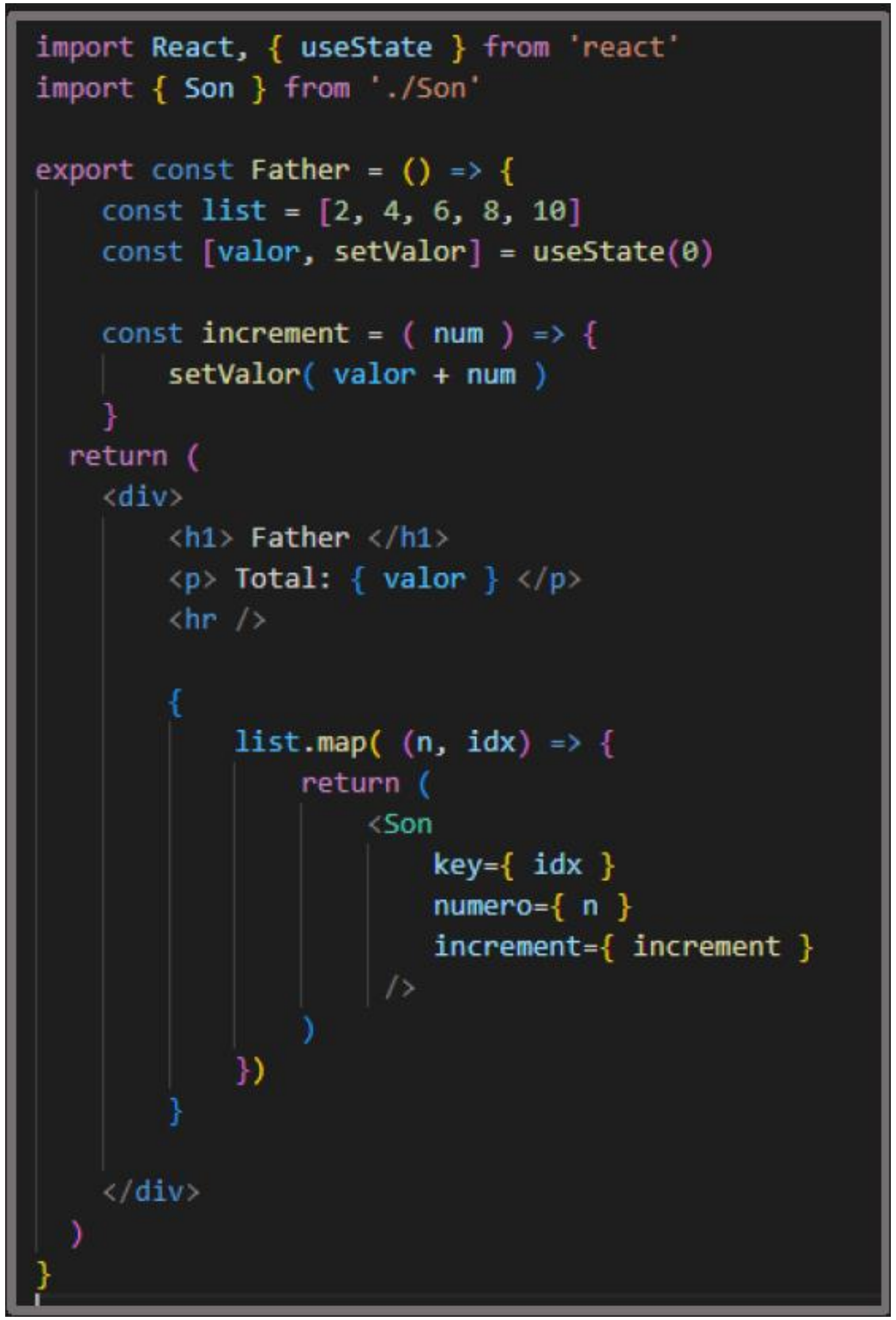


CHALLENGE 05

1. Fix follow components to avoid re renders.
2. After click on each button, <u>*numero*</u> should show the value of <u>*n*</u> on screen



```
import React from 'react'

export const Son = ({ numero, increment }) => {
    console.log('again reloaded...');
  return (
    <button
        className='btn btn-primary mr-3'
        onClick={() => { increment(numero) }}
    >
        { numero }
    </button>
  )
}
```



```
import React, { useState } from 'react'
import { Son } from './Son'

export const Father = () => {
    const list = [2, 4, 6, 8, 10]
    const [valor, setValor] = useState(0)

    const increment = ( num ) => {
        setValor( valor + num )
    }
  return (
    <div>
        <h1> Father </h1>
        <p> Total: { valor } </p>
        <hr />

        {
            list.map( (n, idx) => {
                return (
                    <Son
                        key={ idx }
                        numero={ n }
                        increment={ increment }
                    />
                )
            })
        }

    </div>
  )
}
```