Refs and Keys in React

Introduction to Refs

Ref provides a way **to access DOM nodes or React elements** created in the render method. It is an **attribute which makes it possible to store a reference to particular DOM nodes or React elements.**

According to React.js documentation some of the best cases for using refs are:

- ➢ managing focus
- ➢ text selection
- ➢ media playback
- ➢ triggering animations
- ➢ integrating with third-party DOM libraries

Usually props are the way for parent components to interact with their children. However, in some cases you might need **to modify a child without re-rendering it with new props**. That’s exactly when refs attribute comes to use.

Need of Refs

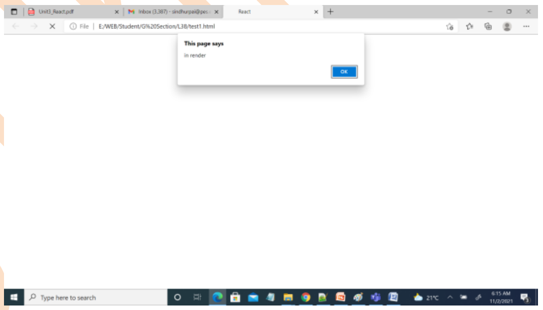
Consider the below code to fulfil the requirement of clicking on the plus button must increment the value of input text box and clicking on the minus button must decrement the value of text box.

Coding Example 1:

```
<div id="root"></div>
        <script type = "text/babel">
        class My_component extends React.Component
        {       constructor()
                {       super();        this.state = {val:0}            }
                render()
                {       alert("in render");
                        return (<div>
                    <button onClick = {this.decrement}>-</button>
                    <input type = "text" value = {this.state.val}/>
                    <button onClick = {this.increment}>+</button>
                    </div>)
                }
                increment=()=>
                {       this.setState({val: this.state.val+1}  )   }
                decrement=()=>
                {       this.setState({val: this.state.val-1})    }
            }
            ReactDOM.render(<My_component/>,
    document.getElementById("root"))
        </script>
```

Outputs:

Case 1: Rendered the page on the browser

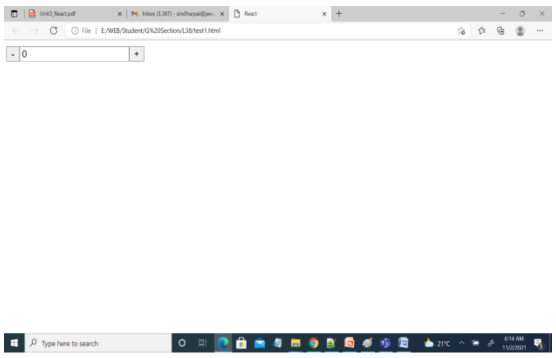

Unit3_React.pdf
Inbox (3,387) - sindhurpai@pes.
React
File
E:/WEB/Student/G%20Section/L38/test1.html

This page says

in render

OK

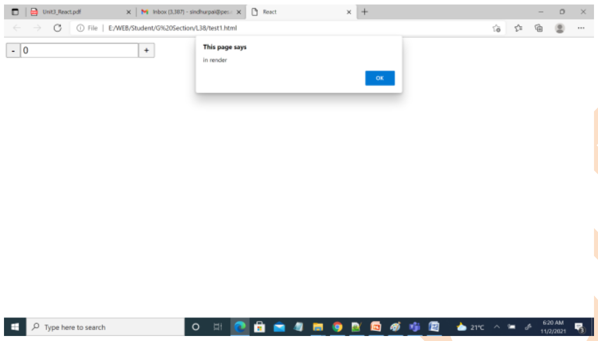
Case 2:OK is clicked


Unit3_React.pdf
Inbox (3,387) - sindhurpai@pes.
React
File
E:/WEB/Student/G%20Section/L38/test1.html
-
0
+

There are two problems in above code.

1. When + buton or – button is clicked, the render function is getting called by default as shown in the output below.
2. Unable to edit the input text box

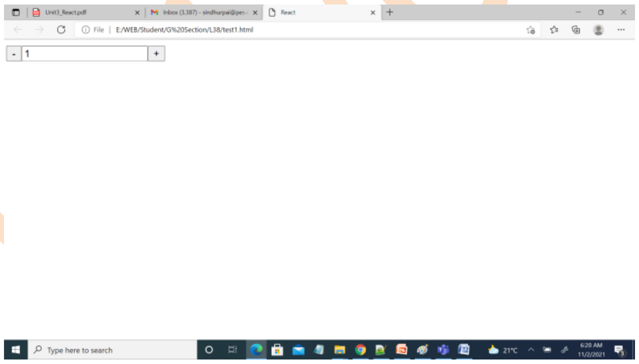
Case 3: + is clicked, observe the mouse pointer


Unit3_React.pdf
Inbox (3,387) - sindhurpai@pes.
React
E:/WEB/Student/G%20Section/L38/test1.html
-
0
+

This page says

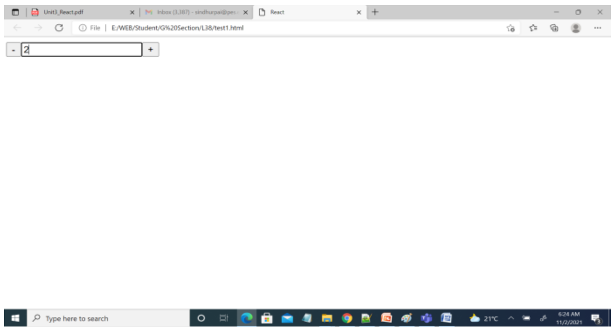
in render

Case 4: Ok is clicked



Note: Same with - button

Case 5:If any key is pressed, input text box no effect



To avoid above problems, we use refs.

Creation and Accessing references

Refs are created using **React.createRef().** Can be assigned to React elements via the **ref attribute**. Refs are commonly assigned to an instance property when a component is constructed so they can be referenced throughout the component.

When a ref is passed to an element in render, **a reference to the node** becomes accessible at the **current attribute of the ref.** The value of the ref differs depending on the type of the node:

When used on a HTML element, the ref created in the constructor with React.createRef() receives the underlying DOM element as its current property.

When used on a custom class component, the ref object receives the mounted instance of the component as its current.

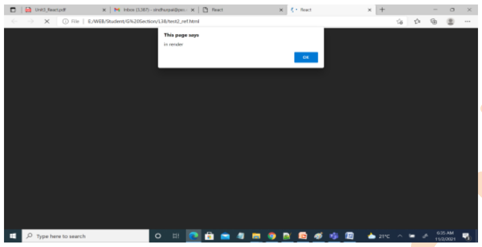
Coding example 2:

```
<div id="root"></div>
        <script type = "text/babel">
                class My_component extends React.Component
                {        constructor()
                        {        super(); this.myref = React.createRef()                }
                        render()
                        {        alert("in render")
                                return (
                                        <input type = "text" ref = {this.myref} />
                                        <button onClick = {this.increment}>+</button>
                                        </div>
                                )        }
                        increment=()=>
                        {        this.myref.current.value++;                        }
                        decrement=()=>
                        {        this.myref.current.value--;                        }
                }
```

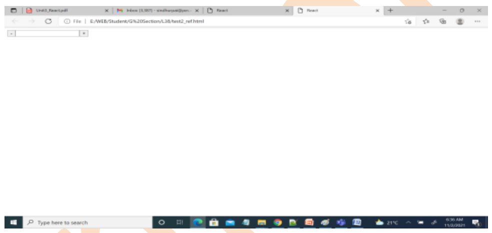
```
            ReactDOM.render(<My_component/>, document.getElementById("root"))
    </script>
```

Outputs:

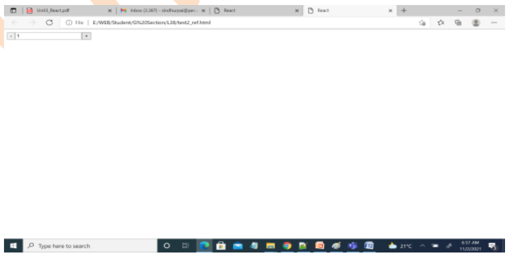
Case 1: When the page is loaded


This page says
in render

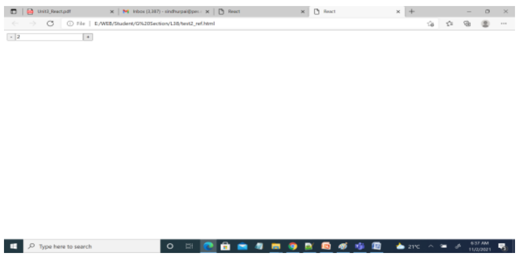
Case 2: When OK is clicked



Case 3: + button is clicked, render not called



Case 4: Again + is clicked, render not called

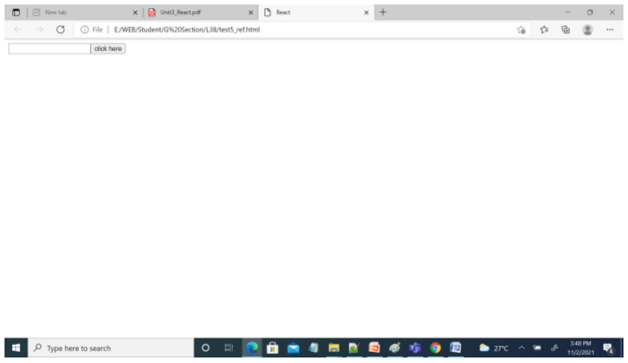


Coding Example 3: Autofocus the input field by adding ref to a DOM element

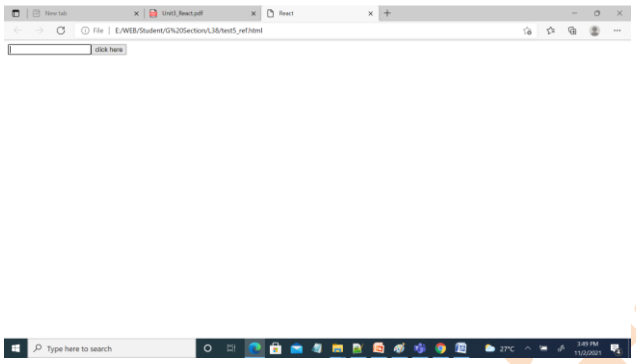
```
        <body>
        <div id="root"></div>
        <script type = "text/babel">
                class CustomInput extends React.Component
                {       constructor()
                        {       super();        this.textref = React.createRef()
                        }
                        focusTextInput=()=>
                        {       this.textref.current.focus()        }
                        render()
                        {       return (<div>
                                        <input type = "text" ref = {this.textref} />
                                        <input type = "button" value = "submit" onClick =
                                        {this.focusTextInput}/>
                                        </div> )
                        }
                }
                ReactDOM.render(<CustomInput />,document.getElementById("root"))
        </script>
</body>
```

Output:

Case 1: When the page is loaded



Case 2: After clicking on click here button



Think about having no button. When the page is loaded, input box must automatically get focused.

Coding Example 4:

```
        <body>
        <div id="root"></div>
        <script type = "text/babel">
                class CustomInput extends React.Component
                {       constructor()
                        {       super(); this.textref = React.createRef()       }
                        componentDidMount()
                         //Removed focusInoutText function. Added the functionality in one of the life
cycle method            {       this.textref.current.focus()                    }
                        render()
                        {       return (
                                        <div>
                                        <input type = "text" ref = {this.textref} />
                                        </div> )
                        }
                }
                ReactDOM.render(<CustomInput />,document.getElementById("root"))
        </script>
 </body>
```

Few points to think:

- Can we have the ref set for component rather than the DOM Element?
- Can you create more than references for the same element?

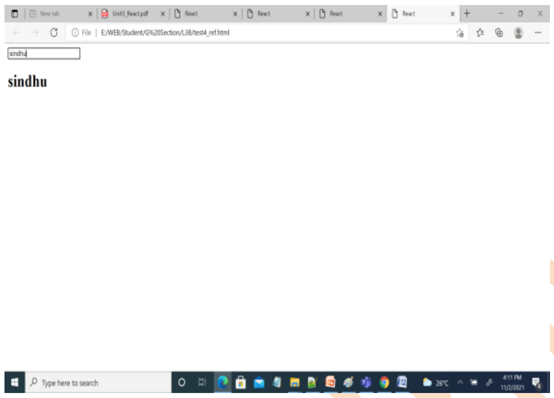
Callback refs

React also supports another way to set refs called "callback refs", which gives more fine-grain control over when refs are set and unset. Instead of passing a ref attribute created by createRef(), you pass a function. **The function receives the React component instance or HTML DOM element as its argument,** which can be stored and accessed elsewhere.

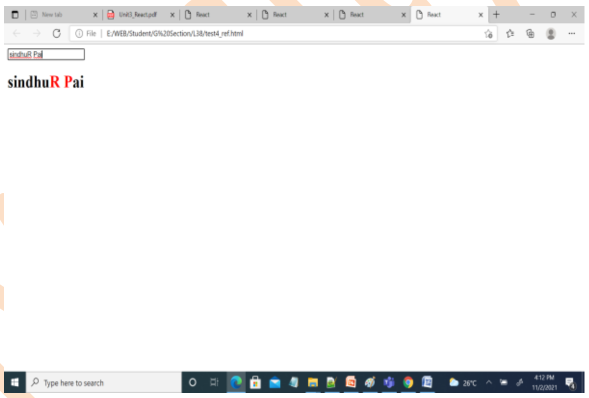
Coding Example 5: Consider the input box. As and when the user types into it, the content of input box is displayed on the page. If the user presses shift key, content has to be displayed in red color

```
 <body>
        <div id="root"></div>
        <script type = "text/babel">
                var txt;
                class My_component extends React.Component
                {       constructor()
                        {  super(); this.myref = (ele) => {this.setref = ele}// callback ref      }
                        render()
                        { return(<div>
                                <input type = "text" onKeyPress = {this.show} />
                                <h1 ref = {this.myref}></h1>
                                </div>
                        ) }
                        show=(e)=>
                        {   txt = e.key
                            if(e.shiftKey){
                              this.setref.innerHTML += '<span style =
                        "color:red"}>'+txt+'</span>'   }
                       else {   this.setref.innerHTML += txt   } // current not available
                    }
               }
               ReactDOM.render(<My_component/>, document.getElementById("root"))
          </script>
     </body>
```

Output:


sindhu

sindhu


sindhuR Pai

sindhuR Pai

Introduction to Keys

A key is a unique identifier which helps to identify which items have changed, added, or removed. Useful when we dynamically created components or when users alter the lists. The best way to pick a key is to choose a string that uniquely identifies the items in the list. Keys used within arrays should be unique among their siblings. However, they **don't need to be globally unique.** Also helps in efficiently updating the DOM.

Coding example 6: Consider an array containing n elements in it. Display these elements using n bullet items in an unordered list

```
<script type = "text/babel">
     const arr = ["book1","book2","book3", ","book4"]
```

```
        function Booklist(props)
                {       return (<ul> <li >{props.books[0]}</li>
                                     <li >{props.books[1]}</li>
                                     <li >{props.books[2]}</li>
                                     <li >{props.books[3]}</li>
                                 </ul>)
                }
        ReactDOM.render(<Booklist books = {arr}/>,document.getElementById("root"))
</script>
```

Observation: As and when the number of elements changes in the array, the code runs into trouble. Refer to the below code to have this dynamism.

Coding example 7:

```
 <script type = “text/babel”>
        function Booklist(props)
        {
                const book_lists= props.books
                //console.log(book_lists)
                const  b  =  book_lists.map((book,index)  =>  <li  key  =  {index}
>{book}</li>)
                return <ul>{b}</ul>
        }
        ReactDOM.render(<Booklist books = {arr}/>,document.getElementById("root"))
</script>
```

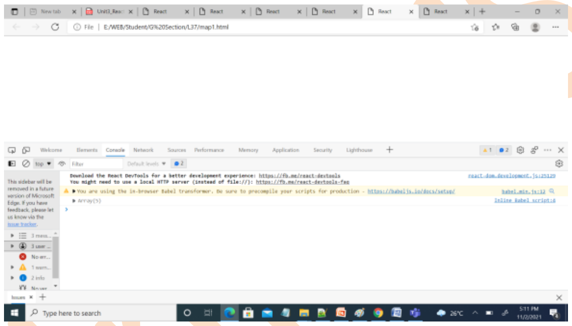
Usage of map

React really simplifies the rendering of lists inside the JSX by supporting the Javascript .map() method. The .map() method in Javascript **iterates through the parent array and calls a function on every element of that array**. **Then it creates a new array with transformed values. It doesn’t change the parent array.**

Coding example 8:

```
<body>
        <div id="root"></div>
        <script type = "text/babel">
                        const numbers = [1, 2, 3, 4, 5];
                        const doubled = numbers.map((number) => number * 2);
                        console.log(doubled); //[2, 4, 6, 8, 10]
        </script>
 </body>
```

Output:

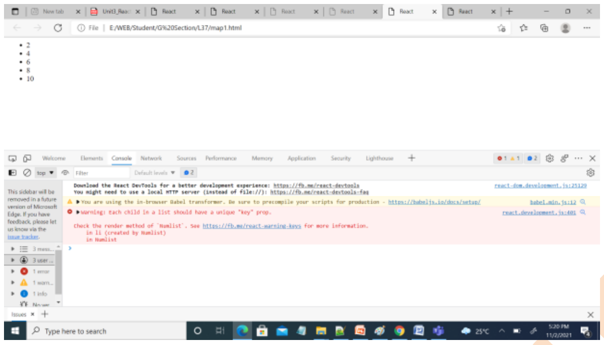


```
Download the React DevTools for a better development experience: https://fb.me/react-devtools     react-dom.development.js:25129
You might need to use a local HTTP server (instead of file://): https://fb.me/react-devtools-faq
▶ You are using the in-browser Babel transformer. Be sure to precompile your scripts for production - https://babeljs.io/docs/setup/     babel.min.js:12
▶ Array(5)     Inline Babel script:4
>
```

Coding example 9: Rendering the doubled numbers on the browser

```
<body>
        <div id="root"></div>
        <script type = "text/babel">
        function Numlist()
        {
                const numbers = [1, 2, 3, 4, 5];
                const doubled = numbers.map((number) => <li>{number * 2}</li>);
                return <ul>{doubled}</ul>
        }
        ReactDOM.render(<Numlist />,document.getElementById("root"))
        </script>
 </body>
```

Output:



```
• 2
• 4
• 6
• 8
• 10
```

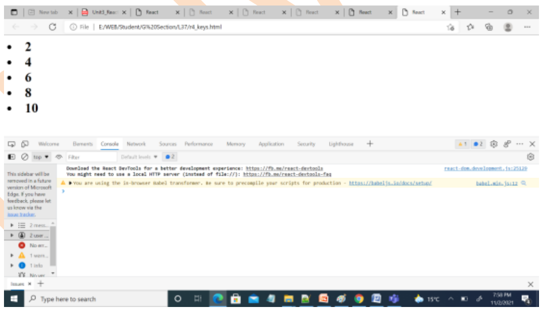
```
Download the React DevTools for a better development experience: https://fb.me/react-devtools        react-dom.development.js:25129
You might need to use a local HTTP server (instead of file://): https://fb.me/react-devtools-faq
▶You are using the in-browser Babel transformer. Be sure to precompile your scripts for production - https://babeljs.io/docs/setup/        babel.min.js:12
▶Warning: Each child in a list should have a unique "key" prop.        react.development.js:401

Check the render method of `Numlist`. See https://fb.me/react-warning-keys for more information.
    in li (created by Numlist)
    in Numlist
```

Note: Please observe the warning. To avoid warning, refer to the below code

Coding example 10:

The below line of statement, **const doubled = numbers.map((number) => <li>{number * 2}</li>);** must be changed to **const doubled = props.numbers.map((number) => <li key = {number.toString()}>{number * 2}</li>);**

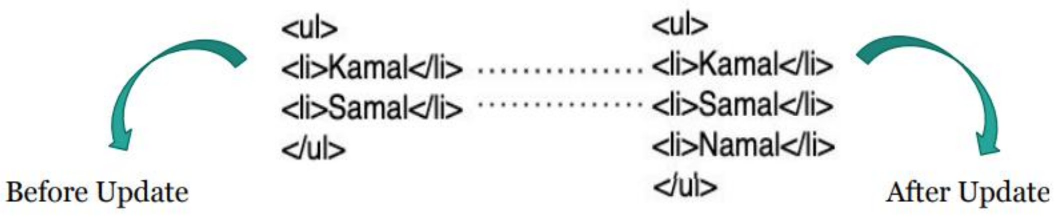


```
• 2
• 4
• 6
• 8
• 10
```

```
Download the React DevTools for a better development experience: https://fb.me/react-devtools        react-dom.development.js:25129
You might need to use a local HTTP server (instead of file://): https://fb.me/react-devtools-faq
▶You are using the in-browser Babel transformer. Be sure to precompile your scripts for production - https://babeljs.io/docs/setup/        babel.min.js:12
```

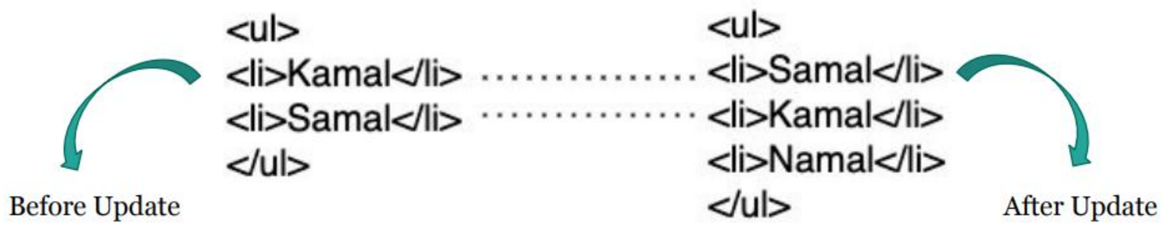
Importance of keys in react

Consider the scenario of updating a list.

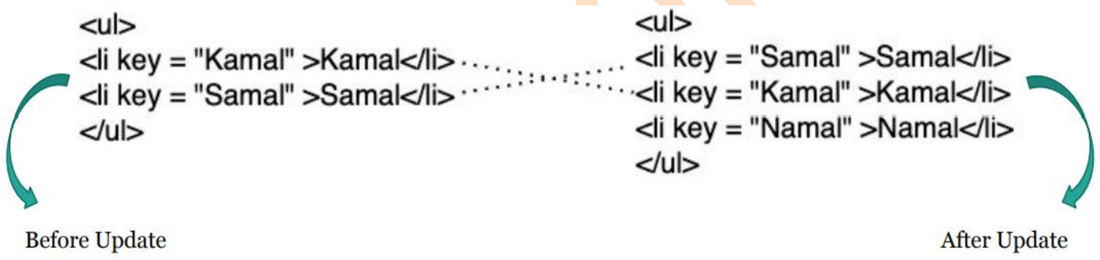


In the above case, React will verify that the first and second elements have not been changed and adds only the last element to the list.

Now, consider a change in the updated list.



In this case, React cannot identify that “Kamal” and “Namal” have not been changed. Therefore, it will update three elements instead of one which will lead to a waste of performance. To solve this, keys are used.



<ul>
<li key = "Kamal" >Kamal</li>
<li key = "Samal" >Samal</li>
</ul>
<ul>
<li key = "Samal" >Samal</li>
<li key = "Kamal" >Kamal</li>
<li key = "Namal" >Namal</li>
</ul>
Before Update
After Update

Try writing the code to update the given list using react keys