**TRAINING**

***2/1/25***

* Functions
* DOM
* Events
* #mini projects
* Major projects
* Html, css, js, React js
* Java, Spring boot, My sql
* Web applications
* Front-end
* Database, java

Git – it is a version controller , it is used to keep track on code

Git-hub – used to store and manage code (cloud storage)

Git-bash –

***3/1/25***

Git is divided into to parts – 1) local repo and cloud

Local repo is divided into to parts – 1) un-tracked files 2) tracked files

Untracked files - create ,update , delete (these will be in local repo).

Command to convert untracked to Tracked files – git add file-name or git add . and git commit -m

Git remote add origin http-path (origin = github) --- It to create

Git push-u origin master/main (u = upstream which means sending byte by byte)

***6/1/25***

**Variables:** are used to store data

1. Static typed and dynamic type:

Case sensitive

Start with letters , \_ , $

Cant start with numbers

Reserved words are not allowed

**Declaration of variables:**

1. Var
2. Let
3. Const

***7/1/25***

Datatypes:

1)Dynamic type language

Two types:

1)Primitive data type

2)Non – primitive data types: Objects, Arrays

**To check the data type -TypeOf:**

**Primitive data type**: Predefined; we can only store the single values

**1)Numbers:** integers, floats

**2) String:** stream of characters enclosed in quotes --- single, double, backtick quotes. Backtick introduced in ES6 version

**3)Boolean:** true or false.

**4)Null:** empty value or no value.

**5) Undefined:** declared variable with no value.

6) **condition:** An expression that evaluates the result (true or false)

7)**conditional Statements:** to make a set of instructions that execute when the condition is true.

**Indentation:** {

//block of code

}

If : only one condition is possible.

***8/1/25***

***JavaScript operators:***

**1)Arithematic operators : + , - , \*,**

Divison - / (Q)

Modulus - % (R)

Exponential - \*\*

**2) Logical operators: && , ||**

**3) Relational Operators: > ,< , <= , >= , == (it checks value), === (it checks datatype) ,!== , !===**

**4) Conditional operator ; Ternary operator(?)**

**Window methods:**

**Prompt:** is used to take input from user. And the default datatype is string

***9/10/25***

**Conditional statements:**

To make the set of instructions execute only when the condition is true.

**Block of code:** set of instructions.

It will execute when the specific condition is true.

Condition: An expression that evaluates a result (True or False)

Ex: Console.log (5>6)

Switch :

Syntax :

Switch(variable/expression){

Case value 1 :

//block of code for case 1

Break;

Case value 2:

//block of code for case 1

Break;

Default:

//body of default

}

No of students = 10

I student = 9

10 \* 9 = 90

***20/01/25***

Loops: Block of code several tyms

1. For loop
2. For in
3. For of
4. While
5. Do while

For loop:

Syntax: for(initialization; condition ; update—expression){

//block of code

}

***21/01/25***

For in loop

Const a = ‘ramya’

For(let I in a){

//block of code

}

While loop

We use while loop when we doesn’t know the no of iterations in advance

Syntax of while loop

While(condition){

//block of code

}

1. Cond inside the ()
2. {}
3. False

Do while syntax:

Do{

//block of code

}while(condition)

Break statement: if it matches with the condition it will stop executing.

Continue statement: if it matches with the condition it will skip that step and execute the remaining program

Nested loops: A loop inside the another loop

The inner loop will execute one time for each iteration of outer loop

An inner loop within the repeating block of outer loop

***23/1/25***

**Functions:**

A function is reusable block of code

It can be called anywhere in the program

You can use the same code with different arguments to get the results

JS:

Keyword = Function

() parameters are the values(variables) which are declared inside the () while defining the function

Arguments: values which are passed through the parameters

**Anonymous function:**

A function without name , After we create a function without name and we assign it to a variable.

Example:

Sum = function(n1,n2){

Let r = n1+n2;

Return r

}

Console.log(sum(8,9))

***24/01/25***

***Arrow functions:***

Es6 version

More readable and more structure

Anonymous functions(lambda functions)

Without function name but they are assigned to a variable

***Synchronous functions and Asynchronous functions***

synchronous programming: operations are performed one after the other, in sequence. So, basically each line of code waits for the previous one to finish before proceeding to the next

Asynchronous functions :Asynchronous programming, on the other hand, allows multiple tasks to run independently of each other.

Promise:

Eventual completion

Three states

1)pending : still not completed---continuing

2)fulfilled == task completed

3)Rejected == The operation is failed

***25/01/25***

***Callbacks:***

A call back function is a function that is passed argument to another function;

Execution: The function receiving the callback will execute the callback at some point during execution (often at the end or aften an asynchronous operation is done)

Example:

function mainFunction (callback){

            console.log("Executing main function: ");

            callback()

         }

         function callbackFunction(){

            console.log("Callback function executed");

         }

**Explanation:**

* mainFunction(callbackFunction);
* mainfunction takes a parameter callback
* callbackFunction is passed as an argument to mainfunction inside the mainfunction, the callback function

**output:**

* mainfunction(callbackFunction);
* it will execute the code inside the mainfunction
* it will print (Executing main function:)
* then it reaches to callback() where it calls the function which is passed to the mainfunction(callbackFunction)

27/01/25

DOM (Document Object Model):

• It is mainly used in web development used to interact and manipulate (change) the html documents.

• When the web page is loaded, the browser creates the DOM of the web page.

• When the html file is loaded into the browser, the JavaScript cannot understand the html document directly.

• DOM: It is basically the representation of the same html document but in a tree-like structure composed of object.

 Document is the entry point of DOM.

Key Points:

• Structure: The DOM represents the structure of html document as a tree. Each node is tree represents the elements, an attribute or text.

• Manipulation: Using JavaScript, you can interact with elements (buttons, paragraphs) on the webpage, changing the content, styles, can ass or remove elements.

• Event Handling: You can handle the interactions like click or pressing the button.

• W3C (World Wide Web Consortium): According to W3C we need to build the webpage.

Methods of Document Object:

Used to interact with and manipulate the document.

1.Getting Element:

• Getting an Element by Id:

• Getting an Element by Class:

• Getting an Element by Tag name:

• Getting an Element by Query Selector

29-01-2025

 Getting an Element by Id:

-It uses the getElementById() method.

 Getting an Element by Class name:

-It uses the getElementsByCLassName() method.

-returns all the elements having the same class name.

-It returns an HTML collection object.

-A HTML Collection is an array of html elements/list of html elements.

 Getting an Element by Tag name:

-Is uses the getElementsByTagName() method.

 Getting an Element by Query Selector:

-The document query selector method selects a HTML elements by tag name, by id, by class name

-QuerySelector(): It can be

-QuerySelectorAll(): It selects all the html elements by its class name and tag name.

Manipulating Elements:

To create HTML elements:

30-01-2025

To create multiple elements:

-Ex1:<script>

const titles=['sai','sam','rahul']

for(let i=0;i<titles.length;i++){

let title=document.createElement("h1")

title.className="a"

title.textContent=titles[i]

console.log(title)

}

</script>

-Ex2:<script>

let title

for(let i=0;i<3;i++){

title=document.createElement("h2")

title.className="tt"

title.textContent=i

console.log(title)

}

</script>

Removing a child element from a parent node:

-Ex:<h1>Removing Child Node</h1>

<ul>

<li>Sai</li>

<li>Sam</li>

<li>Sanvi</li>

</ul>

<script>

const ul=document.querySelector('ul')

const lists=document.querySelectorAll('li')

for(const list of lists){

console.log(Removing ${list.textContent})

ul.removeChild(list)

}

</script>

To append a new element:

-append(): To append new element to the parent node.

-Ex:<div id="a">

<h1>sia</h1>

<h1>ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let n=document.createElement("h2")

n.textContent="Sai is the team leader"

//get the parent element

let p=document.getElementById("b")

console.log(p.innerText)

//To append new element to the parent node

p.appendChild(n)

//log the inner text of the parent

console.log(p)

console.log(p.innerText)

</script>

31-01-2025

InsertBefore():

-It inserts a new node before an existing node or child node of a specific parent node.

-This requires a reference to both and the existing node you want to insert before.

-Ex1:<div id="a">

<h1>sia</h1>

<h1 id="r">ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let n=document.createElement("h2")

n.textContent="Rahul"

let p=document.getElementById('a')

let l=document.getElementById('r')

p.insertBefore(n,l)

console.log(p)

</script>

-Ex2:<div id="a">

<h1 id="c">sia</h1>

<h1 id="r">ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let nc=document.createElement("h1")

nc.innerText="Chandana"

let pc=document.getElementById('a')

let lc=document.getElementById('c')

pc.insertBefore(nc,lc)

console.log(pc)

let n=document.createElement("h2")

n.innerText="Rahul"

let p=document.getElementById('a')

let l=document.getElementById('r')

p.insertBefore(n,l)

console.log(p)

</script>

To remove an element:

-Ex:let b=documnet.getElementById("b")

b.remove()

To remove a child element:

-Ex: :<div id="a">

<h1 id="c">sia</h1>

<h1 id="re">ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let n=document.getElementById("a")

let l=document.getElementById("re")

console.log(l)

n.removeChild(l)

</script>

To replace an child element:

-Ex: let n=document.createElement("h1")

n.innerText="Version it"

let p=document.getElementById("a")

p.replaceChilderen(n)

console.log(p)

03-02-2025

 React.js:

-JavaScript library.

-Single page application.

-Advantages:

 All the updates, components, update specify, update, refresh.

-Example: Amazon

04-02-2025

To create an element by react.js:

-React.createElement(“hi”)

-Syntax:

• React.createElement(type, prop)

-Type: tag name(div,h1,p)

-props: classname, id, onClick.

To display the element:

-ReactDOM.createRoot()

-ReactDOM.render()

-Syntax:

-Reactelemnet: what to render

-Container: where to render

 Intro to JSX:

-JavaScript XML (JSX) is a syntax extension for JS in react.js.

-It allows us to write html code in React.

-Make us easier to write html in react.

o JSX: JSX code gets complied into JS.

o Babel: A tool converts html code into JS.

 While using babel all the html tags must be closed.

With JSX:

-Ex: <div id="app"></div>

<script type="text/babel">

const element=<h1 classname="a">Hello JSX!</h1>

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(element)

</script>

Without JSX:

-Ex:<div id="app"></div>

<script type="text/babel">

const element=React.createElement('h1',{classname:"a"},"Hello JSX!")

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(element)

</script>

Using a call function:

-You can call the function call directly within the JSX expression.

-Ex: <div id="a"></div>

<script type="text/babel">

function getGreeting(name){

return Hello,${name}!Welcome to React

}

function Greeting(){

const name="Sam"

return <h1>{getGreeting(name)}</h1>

}

const root=ReactDOM.createRoot(document.getElementById('a'));

root.render(</Greeting>)

</script>

05-02-2025

Expressions:

Ex: <div id="app"></div>

<script type="text/babel">

const n=10;

const m=<h2>React is {n\*2} times better than jsx</h2>

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(m)

</script>

Conditional Rendering:

Ex:<div id="app"></div>

<script type="text/babel">

const App=()=>{

const isLoggedIn=false;

return(

<div>

{isLoggedIn?(<h1>Welcome back user!</h1>):(<h1>Please log in.</h1>)}

{isLoggedIn &&<h2>You have new notification</h2>}

</div>

);

};

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(<App/>)

</script>

***04/02/25***

React js:

Create a element :

React.createElement(‘h1’)

Syntax:

React.ceateElement(type,prop)

Type = tag name(div, h1, p)

Props = className, id , onClick

--to display the element

---ReactDom.createRoot()

--ReactDom.render() –to display the element

Syntax:

ReactElement – what to render

Container = where to render

Introduction to JSX

It is the javascript XML is sysntax extension for js in react js ; it allows us to write html code in react

Make us easier to write Html in react

JSX: JSX code gets compiled into JS

Babel : a tool converts translates into JS

***05/02/25***

With JSX:

<div id ="app"></div>

    <script type ="text/babel">

        const element = <h1 className ="a"> Hello JSX! </h1>

        const root = ReactDOM.createRoot(document.getElementById('app'));

        root.render(element);

    </script>

Expressions in JSX:

With jsx you can write the expressions inside the { }

This allows us to embed javascript expressions directly ;

***06/02/25***

npx—nope package executor.

Index.html—The main html file that serves as the entry point to the react app

In src folder we have index.js – The javascript entry point for the React app where the dom is rendered

Src/app.js : The main component that serves as the root of the component tree

Components are the core building blocks of a react application. They help to create reusable block of code . If anything goes wrong in the user interface, then it is very easy to find out and solve that.

APP.JS

import logo from './logo.svg';

import './App.css';

import Welcome from './Welcome';

const App=()=>{

  return(

    <div style={{textAlign: 'center' ,color : 'red'}}>

      <Welcome name="Snehitha" greeting ="hello"/>

      <Welcome name="Bhavitha" greeting ="hai"/>

    </div>

  )

}

// function App(){

// // return (

// // <div className ="App">

// // <Welcome/>

// // </div>

// // )

// // }

export default App;

Welcome.js

import React from 'react';

const Welcome=(props)=>{

    const{name,greeting} = props;

    return(

        <h1 className ="h1heading">{greeting},{name}</h1>

    )

}

export default Welcome;

App.css

.message{

  color:black;

  background-color: aquamarine;

  font-size: 40px;

  text-align: center;

**07/02/25**

**In greetings.js**

import React from "react";

function Greeting({isLoggedIn}){

    if(isLoggedIn){

        return <h1>Welcome Back!</h1>

    }else{

        return <h1>Please login</h1>

    }

}

export default Greeting;

**In app.js**

import Greeting from './greetings';

import './App.css';

const App=()=>{

  const isLoggedIn = true;

  return(

    <div className="App">

      <Greeting isLoggedIn = {isLoggedIn}></Greeting>

    </div>

  )

}

export default App;

list and keys in react:

In react the list is the collection of items you want to show on screen

Keys in react are used to keep the track of items. By using these keys react knows which items to chnge, and to make updates.

Map(): elements in array

Function will be applied to all the elements

Const n = [1,2,3,45]

Const d =n.map(num => num \*2)

Console.log(d);

//output: 2,4,8,10

import './App.css';

const FruitList = () =>{

  const fruits = ['apple', 'Mango', 'cherry'];

  return (

    <div>

      <h1>Fruits List</h1>

      <ul>

        {fruits.map((fruits,index)=>(

          <li key={index}>{fruits}</li>

        ))}

      </ul>

    </div>

  )

}

export default FruitList;

State :

State In react:

In react state is a like a container that holds the data or information for a component.

This data can be change over time based on user actions or events. It allows the components to be dynamic, interactive and capable of responding to user input or change over time.

useStateHook:

use state allows you to add state to functional components.

Syntax:

Const[state , setState] = useState(initial value)

State : This is the current state

setState: This function is to upadate the state.

Initial value: the value you want to set as the initial value

Why state is Important:

It allows us the component to remmber things

Ex: if you click a button to change a color ; the state will store hold the color and show on screen

1. functional components

Syntax:

Const[stateVariable, set StateFunction] = useState(initialValue)

1)stateVariable: hold the current state (ex: name)

2)setstateFunction: a function which is used to update the state.

3) initialValue: The iniyial value of the state variabale when the component first renders.

React Hooks:

Introduced on react 16.8 version to manage state and life cycle features in functional components.

**CounterExample:**

import React, {useState} from 'react'

//counter Components

const Counter =()=>{

  const [count, setCount] = useState(0);

  //curent state = count

  //function to update state = setcount

  //useState() to initaliza the state = 0;

  return(

    <div>

      <h1 className='Count'>{count}</h1>

      <button className= 'IncrementButton'onClick={()=>setCount(count+1)}>Increment</button>

      <button className= 'decrementButton'onClick={()=>setCount(count-1)}>Decrement</button>

    </div>

  )

}

export default Counter; //export the component to use it in other files

**ThemeChanger:**

import React,{useState}from 'react';

const ThemeToggler=()=>{

  const[theme,setTheme] = useState('light');

  const toggleTheme=()=>{

    setTheme(prevTheme=>(prevTheme== 'light'?'dark':'light'))

  }

  return(

    <div style={{backgroundColor:theme=='light'?'white':'black' , color:theme=='light'?'black':'white', textAlign:'center' , height : '100vh'}}>

      <h1> The Current theme is {theme}</h1>

      <button onClick={toggleTheme}> toggleTheme</button>

    </div>

  )

}

export default  ThemeToggler;

**Assignment -2**

import React from "react";

const ProfileCard=(props)=>{

    const {name, greeting, Imgsrc} = props;

      return(

        <div className="profile-card">

            <img src={Imgsrc} alt='profile'/>

            <h1>{greeting},{name}</h1>

        </div>

      )

}

export default ProfileCard;

In APP.JS

mport React,{useState}from 'react';

import ProfileCard from './profileCard';

function App(){

  return(

    <div>

      <ProfileCard

      Imgsrc ="https://tse2.mm.bing.net/th?id=OIP.gASMlPqsrIt\_9q8Y76PZKgHaFj&pid=Api&P=0&h=180"

      name ="Snehitha"

      greeting = "hello"

      ></ProfileCard>

    </div>

  )

}

export default App;

React memo:

It is a high order component is not a React hook.

It stops the unessary -rendering of functional components of its props.

It will improve the performance of the functional component.

ChildA.js

const ChildA=()=>{

    console.log("Child A rendered")

    return <h2>This is Child A</h2>

};

export default ChildA;

ChildB.js

const ChildB =({count , increment})=>{

    console.log("Child B rendered")

    return(

        <div>

            <h1>Count:{count}</h1>

            <button onClick={increment}>Increment</button>

        </div>

    )

}

export default ChildB;

App.js

import {useState}from 'react';

import ChildA from './ChildA'

import ChildB from './ChildB'

const Parent =()=>{

  const[count, setCount] =useState(0);

  const increment = ()=>{

   setCount(c=>c+1)

  };

  return (

    <div>

      <ChildA/>

      <ChildB count={count} increment={increment}/>

    </div>

  )

}

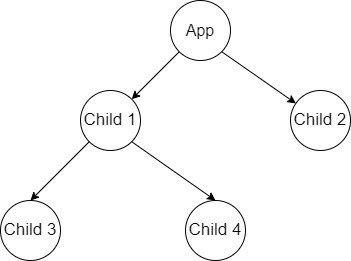
export default Parent;

Whenever we click on increment button both the child A and child B gets rendered because their parent is rendering.

UseContext();

Props(properties) in React are used to pass data from the parent component to child component.

Prop drilling is the process of passing the data from parent component to deeply (nested) child component through intermediate components by passing data as props at each level.



import React from 'react';

function App(){

  const data = "Hello I am Snehitha";

  return(

    <div>

      <User1 data ={data}></User1>

    </div>

  );

};

function User1({data}){

  return <User2 data={data}/>

}

function User2({data}){

  return <User3 data={data}/>

}

function User3({data}){

  return <div>{data}</div>

}

export default App;

React Use Context:

import React, {createContext, useContext} from 'react';

//create the context to hold the data

const DataContext = createContext()

function App(){

    //The data we want to share with user4

    const data = "Hello. I am Snehitha";

    return(

        <DataContext.Provider value={data}>

            <User1/>

        </DataContext.Provider>

    );

};

//user1 component

function User1({data}){

    return<User2/>

};

//user2 component

function User2({data}){

    return<User3/>

};

//user3 component

function User3({data}){

    return<User4/>

};

//user4 component

function User4(){

    const data = useContext(DataContext)

    return <div>{data}</div>

}

export default App;

Assignment

import React, { createContext, useContext } from 'react';

const DataContext = createContext();

function App() {

const data = "Dakshita";

const data1 = "2211cs010138@mallareddyuniversity.ac.in";

return (

<DataContext.Provider value={{ data, data1 }}>

<Profile />

</DataContext.Provider>

);

}

function Profile() {

const { data, data1 } = useContext(DataContext);

return (

<div>

<h1>{data}</h1>

<p>{data1}</p>

</div>

);

}

export default App;

React Hooks

React Reducer();

Callback()

React Reducer();

A reducer is a function that helps to update (like score, a list) based on action we take

---current state : The current value of a state

---Action :

Example : add or remove an item

State : initial value (3 items)

--Tracking apples :

import React, {useReducer} from 'react'

///reducer function to track the number of apples

function appleReducer(state=3,action){

    switch(action.type){

        case 'ADD\_APPLE':

            return state+1 ;

        case 'EAT\_APPLE':

            return state-1;

        default:

            return state;

    }

}

function App(){

    const[state,dispatch]=useReducer(appleReducer , 3)

    return(

        <div style={{padding:'20px', display: 'flex', flexDirection : 'column' , alignItems: 'center' , justifyContent : 'center'}}>

            <h1>Apple Counter</h1>

            <p>Number of apples:{state}</p>

            <button style ={{

            backgroundColor:'peachpuff',

             border:'none' ,

             borderRadius:'5px' ,

             height:'30px' ,

             margin: '10px',

             cursor: 'pointer'

            }}

            onClick={()=>dispatch({type : 'ADD\_APPLE'})}> ADD APPLE</button>

            <button style ={{

            backgroundColor:'peachpuff',

             border:'none' ,

             borderRadius:'5px',

             height:'30px' ,

             margin: '10px',

             cursor: 'pointer',

              }}

               onClick={()=>dispatch({type : 'EAT\_APPLE'})}> EAT APPLE</button>

        </div>

    )

}

export default App;

Callback in React :

--in React , a call back is function that is passed as on argument to another function

--call backs are often to communicate with different components

Ex : A parent component passing as callback to a chid

--parent component displays a message ; and child component has a button

When the button is clicked , the child will use callback and tells the parent to update

App.js

import React,{useState} from 'react';

import Child from './Child';

const   Parent =()=>{

    const [message, setMessage] = useState('Hello World')

    const changeMessage=()=>{

        setMessage("you clicked the button")

    };

    return(

        <div>

            <h1>{message}</h1>

            {/\* passing the change message function as  \*/}

            <Child changeMessage={changeMessage}></Child>

        </div>

    )

}

export default  Parent;

Child.js

import React from 'react';

const Child =({changeMessage})=>{

    return <button onClick={changeMessage}> Click me</button>

}

export default Child;

import React, { useReducer } from 'react';

// Reducer function to track the number of apples

function appleReducer(state = 3, action) {

  switch (action.type) {

    case 'ADD\_APPLE':

      return state + 1;

    case 'Eat\_APPLE':

      return state - 1;

    default:

      return state;

  }

}

function App() {

  const [count, dispatch] = useReducer(appleReducer, 3);

  return (

    <div style={{ textAlign: 'center', padding: '50px', fontSize: '20px', margin: 'auto', width: '50%', borderRadius: '10px', marginTop: '50px' }}>

      {Array.from({ length: count }).map((\_, index) => (

        <img key={index} src={require('./Apple.jpg')} alt='Apple' style={{ width: '100px', height: '100px', margin: '5px' }} />

      ))}

      <h1>Apple Counter</h1>

      <p>Number of apples: {count}</p>

      <button

        style={{ margin: '10px', padding: '10px 20px', fontSize: '16px', backgroundColor: 'pink', color: '', border: 'none', borderRadius: '5px' }}

        onClick={() => dispatch({ type: 'ADD\_APPLE' })}

      >

        Add Apple

      </button>

      <button

        style={{ margin: '10px', padding: '10px 20px', fontSize: '16px', backgroundColor: 'pink', color: 'black', border: 'none', borderRadius: '5px' }}

        onClick={() => dispatch({ type: 'Eat\_APPLE' })}

      >

        Eat Apple

      </button>

    </div>

  );

}

export default App;

useRef:

• useState: changes the state, React will render the component and update the UI with the new element.

• useRef: changing the value by using useRef does not re-render; stays the same even when the value is updated.

• useRef: when you change the value stored in ref.current, it does not make the componentrender. This is useful when you need to store values or interact with DOM elements without extra re-renders.

• Commonly used:

o Accessing a DOM element: like scrolling to screen

o Focusing the input

o Storing values: It will keep track of values that don’t need to update the UI but need to remember the previous state.

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

function CounterWithPrev(){

    const [count, setCount] = useState(0);

    const prevCount = useRef();

    //create a ref to store the previous value

    useEffect(()=>{

        prevCount.current = count;

    }, [count])

    return(

        <div>

            <p>Current Counter: {count}</p>

            <p>prevCount: {prevCount.current}</p>

            <button onClick={()=>setCount(count+1)}>Increment</button>

        </div>

    )

}

export default CounterWithPrev;