Unit-I : HTML 5

Semantic Element

A semantic element clearly describes its meaning to both the browser and the developer

The example of the non-semantic elements are

1. , 2. they tells nothing about its contents

The examples of semantic elements are

1. <form> 2. <table> 3. <article> they clearly define its content

SVG and Canvas

Canvas	SVG
1. Pixel Graphics	1. Vector Graphics
2. Resolution dependent	2. Resolution independent
3. Drawn with Javascript	3. XML format
4. modified through script only	4. Modified through CSS & script
5. not suitable for printing at high resolution	5. High quality print at any resolution
6. Suitable for games	6. Not suitable for games

Web-storage

web storage API provides mechanisms by which browsers can store key/value pairs in a much better way than using

Unit-I HTML 5

Semantic Element

A semantic element clearly describes its meaning to both the browser and the developer

The example of the non-semantic elements are

1. , 2. they tells nothing about its contents

The examples of semantic elements are

1. <form> 2. <table> 3. <article> they clearly define its content

SVG and Canvas

Canvas	SVG
1. Pixel Graphics	1. Vector Graphics
2. Resolution dependent	2. Resolution Independent
3. Drawn with Javascript	3. XML format
4. modified through script only	4. Modified through css & script
5. not suitable for printing at high resolution	5. High quality print at any resolution
6. Suitable for games	6. Not suitable for games

Web-storage

web storage API provides mechanisms by which browsers can store key/value pairs in a much better way than using cookies, the two mechanisms within web storage are as follows

i) Session storage: maintainance a seperate area for each given origin thats available for the duration of page session stores data only for a session meaning that data is stored until the browser is closed.

ii) local storage: does the same thing but persists even the browser is closed and reopened. Stores data with no expiration date and gets clear only through javascript or cleaning the browser cache/locally stored data.

storage limit is larger than the cookie and same as session storage upto 5MB

web worker and service worker

web worker:

web workers makes it possible to run a script operation in a background thread seperate from the main execution thread of web application. Advantage of this is that laborious processing can be performed in seperate thread along the main (usually the UI) thread to run without being blocked/ slowed down

Service worker:

Service worker essentially acts as proxy servers that sit between web applications, the browser and network, they are intended to enable the creation of effective offline experiences, intercept network requests.

Geolocation API:

The Geolocation API allows us to get the physical location (geographical position) of any device when given permission by the user. It is a well-supported API implemented in more than 90% of desktop and mobile browsers. We can get the geolocation information of device by using the navigator (GPS). It gives us information like latitude, decimal degrees, the longitude in decimal degrees, the altitude in meters, the direction the device is heading towards the speed the device is travelling, and accuracy of latitude & longitude measured in meters.

The Geolocation API is accessed via a call to navigator.geolocation; this will cause the user's browser to ask them for permission to access their location data.

- Geolocation.getCurrentPosition() : Retrieves the device current location.

- geolocation.watchPosition() : Registers a handler function that will be called automatically each time the position of device changes, returning the updated location.

The getCurrentPosition() method returns an object on success. The latitude, longitude and accuracy properties are always returned.

Javascript:

Declaring a variable

before ES6 (ECMA 2015)

1. var keyword

```
var <variable-name>;
```

After ES6 (ECMA 2015)

1. var
2. let
3. const

Declaring a function:

```
function sum(a,b) {
    return a+b;
}
```

How to get input

1. form
2. Predefined object window we have prompt method

```
window.prompt("string")
```

Conversion functions

```
parseInt("10.5") → 10
parseFloat();
```

Displaying output

```
document.write()
document.writeln()
```

to access the HTML form elements in javascript

id:

document.getElementById("no1").value = 10;

accessing the value:

document.getElementById("no1").value

name:

document.getElementByName("no1").value = 10;

access:

document.getElementByName("no1").value

```
<!DOCTYPE html>
<html>
 <head>
<title> Java script Arithmetic Operators Demonstration
 </title>
<script>
    function operation(opcode) {
        var a = parseInt(document.getElementById('n1')
                                                .value);
        var b = parseInt(document.getElementById('n2').
                                                value);
        var c = document.getElementById('res');
        if( opcode == 'ADD') {
              c.value = (a+b);
         }
         else if (opcode == 'SUB') {
              c.value = (a-b);
          }
         else if (opcode == 'MUL') {
              c.value = (a*b);
         }
         else if (opcode == 'DIV') {
              c.value = Math.floor(a/b);
         }
         else { c.value = "Invalid code";
         }
```

```
    }
  </scalpt>
</head>
<body>
<table>
 <tr>
 <th> Enter first number </th>
 <td><input type = "text" id = "n1" /></td></tr>
 <tr> <th> Enter second number </th>
      <td><input type = "text" id = "n2" /></td></tr>
 <tr><th> Result </th>
      <td><input type = "text" id = "res" /> </td></tr>
 <tr> <td>
      <input type = "button" value = "ADD"
       onclick = " operation('ADD') "/>    
      <input type = "button" value = "SUB"
       onclick = " operation('SUB') "/> </td>
      <td>
       <input type = "button" value = "MUL"
        onclick = " operation('MUL') "/>    
      <input type = "button" value = "DIV"
       onclick = " operation('DIV') "/>
      </td> </tr> </table>
</body>
</html>
```

(2)

```
<!DOCTYPE html>
<html>
<head>
<title> Denominations </title>
</head>
<body>
<script>
var amount = parseInt (prompt ("Enter Amount"));
var twoth = 0, fivhun = 0, twohun = 0, hun = 0, fifty = 0,
 twenty = 0, ten = 0, fiv = 0, two = 0, one = 0;
```

```
if( amount>2000) {
    twoth = Math.floor(amount/2000);
    amount = amount %.2000;
}
if(amount >500) {
    fivhun = Math.floor(amount/ 500);
    amount = amount %.500;
}
if(amount 7200) {
```

③ write a javascript program to add 3 buttons to set the background color & 3 buttons to set text color.

mime type

24 position: Relative;

- An element with position: relative; is positioned relative to its normal position.
- Setting the top, right, bottom and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

```
div.relative {
    position: relative;
    left: 30px;
    border: 3px solid #73AD21;
}
```

Example for position property

```
<!DOCTYPE html>
<html>
<head>
<title> CSS Position property </title>
<style>
.parent {
    display: inline-block;
    border: 2px dotted black;
    height: 1500 px;
}
.box {
    display: inline-block;
    width: 150 px;
    height: 150 px;
    background: yellow;
}
#two {
    background: green;
    position: relative;
    left: 20px;
    top: 20 px;
    z-index: -1;
}
```

```
</style>
</head>
<body>
<div class="parent">
    <div class="box" id="one"></div>
    <div class="box" id="two"></div>
    <div class="box" id="three"></div>
</div>
</body>
</html>
```

1Q) Write a javascript program to generate 10 multiples of number

2Q) Write a javascript program to check whether the number is armstrong or not

3Q) Write a javascript program to check whether the number is perfect or not

4Q) Write a javascript program to ~~accept~~ check generate fibonacci series

5Q) Date object → explore.

6Q) Design a tribute page to Virat Kohli. The webpage should display his image and his acheivements including the following:

- Two buttons with labels: Records as batsman, Remarks on captaincy
- Dropdown list box with 3 options: IPL, ODI and Test-Cricket

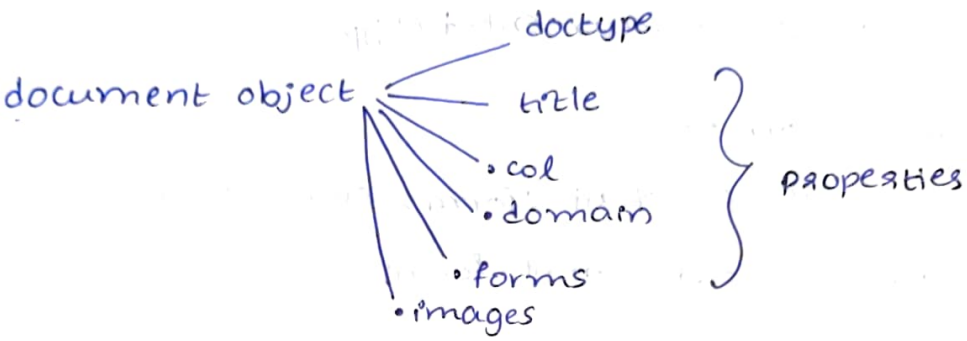
Sample text (need not be a historical fact) displaying two of his acheivements in IPL and ODI

7/8/24

Methods of Math object

1) floor()
2) ceil()
3) round()
4) min()
5) pow()
6) random()
7) sign()
8) sqrt()
9) trunc()

Date object & Array object.



methods:

getElementById()
getElementByClassName()
getAttribute()
innerHTML()
querySelector()
createElement()

Document object properties

1. document.title :-

- Gets or sets the title of document

syntax:

```
Let title = document.title;
document.title = 'New title';
```

2. document.url :-

- Gets or sets the title of document

syntax:

```
let title = document.title;
document.title = 'New Title';
```

```
let url = document.URL;
```

3. document.domain:

- Gets or sets domain of document

Syntax:

```
let domain = document.domain;
document.domain = 'new domain.com';
```

4. document.doctype:

- Returns the document doctype

Syntax:

```
let doctype = document.doctype;
```

5. document.forms:

- Returns a collection of all <form> element in doc

Syntax: let forms = document.forms;

6. document.images:

- Returns a collection of all <img> elements in the document.

Syntax: let image = document.images;

Document object Model methods:

1. getElementById()

Finds an element by its ID

Syntax:

```
let element = document.getElementById('elementID');
```

2. getElementByClassName():

Finds all elements with the specified class names

syntax:

```
let elements = document.getElementByClassName
('className');
```

3. createElement():

creates a new HTML element

Syntax: let newelement = document.createElement
('tagname');

4. GetAttribute() :

- Retrieves the value of specified attribute an element

syntax:

```
let value = element.getAttribute('attributeName');
```

5. innerHTML :

- Gets or sets the HTML content of an element

syntax: element.innerHTML = 'new HTML content';

6. querySelector()

Returns the first element that matches specified CSS selector

Syntax:

```
let element = document.querySelector('selector');
```

13/8

Canvas

canvas is an HTML element which is used to draw the graphics and animations by using scripting API or WebGL API

1. creating canvas element

```
<canvas id="mycanvas"></canvas>
```

default width: 300 px
default height: 150 px

2. Accessing the canvas element in javascript

```
var canvas = document.getElementById('mycanvas');
```

3. check if the browser supports the canvas element and getting the context of canvas object

```
if (canvas.getContext())
{
    var ctx = canvas.getContext('2D');
}
else {
    document.write("Your browser does not support the canvas element.");
}
```

4. drawing the graphics

```
ctx.fillStyle = "red";
ctx.fillRect(50,50,100,50);
```

Program [Canvas]

```
<!DOCTYPE html>
<html><head>
<meta charset="UTF-8"><style>
/* Add some styles if needed */
canvas {
    border: 1px solid #000;
}
</style></head>
<body>
<h1> Canvas Example </h1>
<canvas id="myCanvas" width="400" height="200">
</canvas>
<script>
var canvas = document.getElementById("myCanvas");
// Check if the browser supports the canvas element
if( canvas.getContext) {
    // Get the 2D rendering context
    var ctx = canvas.getContext("2d");
    // Draw a red rectangle
    ctx.strokeStyle = "red";
    ctx.strokeRect(50, 50, 100, 50);
    // Draw a blue circle
    ctx.beginPath();
    ctx.arc(300, 100, 30, 0, 2 * Math.PI);
    ctx.fillStyle = "blue";
    ctx.fill();
    ctx.closePath();
} else {
    document.write("Your browser does not support thi
```

```
canvas element.");
}
</script>
</body>
</html>
```

SVG

Program:

```
<!DOCTYPE html>
<html><head><title>
Canvas and SVG </title></head>
<body>
<svg width="100" height="100">
<circle cx="50" cy="50" r="40" stroke="green"
stroke-width="20" fill="yellow" /></svg>
</body>
</html>
```

Web storage

```
setItem(key,value);
getItem(key);
removeItem(key);
clear();
```

The above methods are available in both local storage and session storage

Q) Design HTML page to demonstrate session storage.

A)

```
<!DOCTYPE html>
<html>
<head>
<title> Demonstration of session storage </title>
<script>
function clickCounter() {
    if( typeof( Storage) !== "undefined") {
        if( sessionStorage.clickcount) {
```

```
        sessionStorage.clickcount = Number(sessionStorage.
                                    clickcount)+1;
    }
    else {
        sessionStorage.clickcount = 1;
    }
    document.getElementById("result").innerHTML =
        "You have clicked the button" + sessionStorage.clickcount
            + " time(s) in this session.";
} else {
    document.getElementById("result").innerHTML =
"Sorry, your browser doesnot support web storage...";
}
}
</script></head><body>
<p><button onclick="clickCounter()" type="button">
Click Me! </button></p>
<div id="result"></div>
<p>Click the button to see the counter increase.</p>
<p>Close the browser tab (or window), and try again,
and the counter is reset.</p>
</body>
</html>
```

First-class Functions

A programming language is said to have First-class functions when functions in that language are treated like any other variable. For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as value to a variable

Example

1) Assigning a function to a variable

```
const display = () => {
        console.log(" Assigning function to variable");
};
display();
```

2) Passing a function as an argument

```
function sayHello() {
    return "Hello, ";
}
function greeting(helloMessage, name) {
    console.log(helloMessage()+name);
}
greeting(sayHello, "Javascript!");
```

3) Returning a function

```
function sayHello() {
  return () => {
        console.log("Hello!");
  };
}
```

High Order Function

A higher order function is a function that accepts functions as parameters and/or returns a function

The most important higher-order functions are

1) map()
2) filter()
3) reduce()

1) .map() method

The .map() method executes a callback function on each element in an array. It returns a new array made up of the return values from the callback function.

The original array does not get altered, and the returned array may contain different elements than the original array.

Ex

```
var x = [1,2,3,4,5]
var y = x.map((n) => n*n);
console.log(x);
console.log(y);
```

O/P:-

```
[1,2,3,4,5]
[1,4,9,16,25]
```

2) .filter() method

The .filter() method executes a callback function on each element in an array. The callback function for each of the elements must return either true or false. the returned array is a new array with any elements for which the callback function returns true.

In above code example, ~~the array filteredArray will contain all the elements of randomNumbers~~.

Ex:

```
var a = [2,3,4,5,6];
Vaa res = a.filter((n) => n%2==0);
console.log(res);
```

O/P:

```
[2,4,6]
```

3) .reduce() method

The .reduce() method iterates through an array and returns a single value.

Ex:

```
var a = [2,3,4,5,6)
var sumofarr = a.reduce((accumulator, currentValue)
                => accumulator + currentValue);
console.log(sumofarr);
```

O/P:-

20

In above example, reduce() method will sum up all the elements of array. It takes a callback function with two parameters (accumulator, currentValue) as arguments. on each iteration, accumulator is the value returned by last iteration and the currentValue is the current element.

Spread operater

ES6 spread operator is used to extract the elements from a collection like array

Ex:

```
> let a = [1, 2, 3, 4];
  let b = [5, 6, 7, 8];
  let c = [9, 10, 11, 12];
  let d = [];
  for (let i=0; i<a.length; i++)
      d[i] = a[i];
  for (let i=0; i<b.length; i++)
      d[i+4] = b[i];
  for (let i=0; i<c.length; i++)
      d[i+8] = c[i];
> d
(12) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Traditional approach

```
> let e = [...a, ...b, ...c];
> e
(12) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

spread operator.

ES6 classes

Creating Box class in Java

```
class Box {
    int l, b, h;
    Box (int x, int y, int z) {
        l = x
        b = y
        h = z
    }
    long volume() {
        return l*b*h;
    }
}
public class BoxDemo {
    public static void main (String args[]) {
        Box b = new Box (10, 20, 30);
        System.out.println ("Volume is " +
                            b.volume());
    }
}
```

Creating a ES6 class in Javascript

```
class Box
{
    constructor (x, y, z) {
        this.l = x
        this.b = y
        this.h = z
    }
    volume()
        return this.l * this.b * this.h;
    }
}
const b = new Box (10,20,30);
Console.log ("Volume of Box is : " + b.volume());
```

command for creating react application

```
npx create-react-app demoapp
```

command for executing your reactapp

```
npm start
```

code . to open in new window in vscode

In react we have component

component have 2 ways:

1) class based component → create using ES6 code
2) functional component
   → creating using normal arrow function.

ES6 Modules

Example for named exports

```
export const name = "Jesse";      // named exports in in-line way
export const age = 40;
```

(or)

```
const name = "Jesse";             // named exports at the bottom of module or file
const age = 40;
export {name, age};
```

Usage of above module

```
import {name, age} from './person';
console.log(name);
consde.log(age);
```

Design a react code to display HelloWorld message

HelloWorld.js

```
import React from 'react';
class HelloWorld extends React.Component
{
    render()
    {
        return <h1> HelloWorld </h1>;
    }
}
export default HelloWorld;
```

App.js

```
//import './App.css';
import HelloWorld from './HelloWorld';
function App() {
    return (
        <div classname = "App">
        <HelloWorld />
        </div>
    );
}
export default App;
```

Design a class based component to implement Counter

Counter.js

```
import React, { Component } from 'react';
class Counter extends Component
{
    constructor (props) {
        super(props);
        this.state = { count : 0, }
    }
    incrementHandler = () => {
```

```
        if(this.state.count < 20)
            this.setState({count: this.state.count+1});
    };
    decrementHandler = () => {
        if(this.state.count > 1)
            this.setState({count: this.state.count-1});
    };
    render() {
        return (
            <div>
                <h1> {this.state.count} </h1>
                <button onclick={this.incrementHandler}>
                 Increment </button>
                        
                <button onclick={this.decrementHandler}>
                 Decrement </button>
            </div>
        )
    }
}
export default Counter;
```

App.js

```
import './App.css'
import Counter from './Counter';
function App() {
    return (
      <div className="App">
      <Counter />
      </div>
    );
}
export default App;
```

TodoList

TodoListComponent.js

```
impost React, {Component} from 'react';
class TodoListComponent extends Component {
    constructor(props) {
        super(props);
        this.state = {
            todos: [],
            newTodo: '',
        };
    }
    handleInput = (event) => {
        this.setState({newTodo: event.target.value});
    };
    addTodo = () => {
        const {todos, newTodo} = this.state;
```

```
    this.setState({
      todos: [...todos, newTodo],
      newTodo: ''
    });
  };

  removeTodo: (index) => {
    const {todos} = this.state;
    this.setState({
      todos: todos.filter((_, i) => i !== index)
    })
  };

  render() {
    const {todos, newTodo} = this.state;
    return (
      <div>
        <h2> Todo List</h2>
        <input type="text" value={newTodo}
onChange={this.handleInput} placeholder="Add new Task"/>
        <button onClick={this.addTodo}>Add</button>
        <ul>
          {
            todos.map((todo, index) => (
            <li key={index}>{todo}
            <button onClick={() => this.removeTodo
            (index)}> remove </button>
            </li>
          ))
          }
        </ul>
      </div>
    )
  }

}
export default TodoListComponent;
```

React Life cycle Methods

The life cycle of react component can be divided into three phases:

Mounting : when the component is being inserted into the DOM

Updating : when the component is being re-rendered due to changes in props or state.

Unmounting : When the component is being removed from the DOM

Mounting
-- constructor
-- getDerivedStateFromProps
-- render
-- ComponentDidMount.

Updating
-- getDerivedStateFromProps
-- render
-- getSnapshotBeforeUpdate
-- componentDidUpdate

Unmounting
-- componentWillUnmount

DigitalClock.js

```
import React, {Component} from 'react';
Import './DigitalClock.css';
class Digitalclock extends component {
    constructor(props) {
        super(props);
        this.state = {
            time: new Date()
        };
    }
    componentDidMount() {
        this.timerID = setInterval(() => this.tick(), 1000);
    }
```

```
componentWillUnmount() {
    clearInterval(this.timerID);
}
tick() {
  this.setState({time: new Date()});
}
render() {
  const {time} = this.state;
  const hours =
  String(time.getHours()).padStart(2, '0');
  const minutes =
  String(time.getMinutes()).padStart(2,'0');
  const seconds =
  String(time.getSeconds()).padStart(2,'0');
  return (
    <div className="digital-clock">
      <div className="digit">{hours[0]}</div>
      <div className="digit">{hours[1]}</div>
      <div className="colon">:</div>
      <div className="digit">{minutes[0]}</div>
      <div className="digit">{minutes[1]}</div>
      <div className="colon">:</div>
      <div className="digit">{seconds[0]}</div>
      <div className="digit">{seconds[1]}</div>
    </div>
  );
}
}
export default DigitalClock;
```

DigitalClock.css

```
.digital-clock {
    display: flex;
    font-size: 5rem;
    font-family: 'courier', monospace;
    color: red;
    background-color: black;
```

```
    padding: 20 px;
    border-radius: 10 px;
}
.digit {
    width: 1em;
    margin: 0 5 px;
    text-align : center;
    background: □ black;
    color : □ #ff0000;
    border : 2 px solid □#ff0000;
    display : inline-block;
}
.colon {
    width : 0.5em;
    margin : 0 5px;
    text-align : center;
    color : □ #ff0000;
}
```

Functional Form of setState() Method

```
this.setState((prevState, props) => {
    return {count: prevState.count + 1};
});
```

2 forms of setState() → Object form
→ functional form

TrafficLight.js

```
import React, {Component} from 'react';
import './TrafficLight.css';
class TrafficLight extends Component
{
    constructor (props)
    {
        super (props);
        this.state = { currentlight = 'red';};
    }
```

```
componentDidMount()
{
  this.startTrafficLightCycle();
}
componentWillUnmount()
{
  clearTimeout(this.lightTimeout);
}
startTrafficLightCycle = () => {
      this.changeLight('red', 4000);
};
changeLight = (nextLight, duration) => {
    this.setState({currentLight: nextLight});
    this.LightTimeOut = setTimeout(() => {
      if(nextLight === 'red'){
          this.changeLight('green', 3000);
      }
      else if (nextLight === 'green'){
          this.changeLight('yellow', 500);
      }
      else if (nextLight === 'yellow'){
          this.changeLight('red', 4000);
      }, duration);
    };
render(){
    const {currentLight} = this.State;
    return (
      <div className='traffic_light'>
      <div className={'lightred ${
currentLight === 'red' ? active : ''}'} ></div>
        <div className={'light green ${
currentLight === 'green' ? active : ''}'} ></div>
        <div className={'light yellow ${
currentLight === 'yellow' ? active: ''}'} ></div>
    );
```

```
  }
}
export default TrafficLight;
```

Trafficlight.css :-

```
.traffic_light {
    width : 80px;
    background-color : #333;
    padding : 20 px;
    border-radius: 10 px;
    display: flex;
    flex-direction : column;
    justify-content : space-around;
    align-items : center;
    height : 220px;
    margin : 0 auto;
}
.light {
    width : 60 px;
    height : 60 px;
    border-radius : 50 %;
    background-color : #555;
    transition : background-color 0.55 case;
}
.red.active {
    background-color : red;
}
.yellow.active {
    background-color : yellow;
}
.green.active {
    background-color : green;
}
```

Q) 27/9/24
With the help of a example explain the usage of ~~class~~ props in class component.

A)

App.js

```
import './App.css';
import HelloWorld from './HelloWorld';
function App() {
    return (
        <div className="App">
        <HelloWorld name='VVIT' branch='CSE'/>
        </div>
    );
}
export default App;
```

HelloWorld.js:

```
import React, {Component} from 'react';
class HelloWorld extends Component
{
    render() {
        const {name, branch} = this.props;
        return <h1>{"Iam pursuing B-Tech ${branch}
at ${name} `} </h1>;
    }
}
export default HelloWorld;
```

1/10/24 Props

Inorder to work Props in class component we have two steps:

1. How to pass props from the parent
   <Child fname="Hello" lname="VVIT" />
2. How to access props in class component
   this.props
   const {fname, lname} = this.props;

Parent.jsx

```
import React, {Component} from 'react';
import child from './child';
class Parent extends component {
    render() {
     return <child fname = "Hello" lname = "VVIT" />
    }
}
export default Parent;
```

Child.jsx

```
import React, {Component} from 'react';
class child extends Component {
   render() {
      const { fname, lname} = this.props;
      return (
        <div>
          <h3>FirstName: {fname} </h3>
          <h3> LastName: {lname} </h3>
        </div>
      )
   }
}
export default Child;
```

Default Props

```
import React, {Component} from 'react';
class Child extends Component {
      static defaultProps = {
           fname: 'Mohan',
           lname: 'krishna',
       };
       render() {
           const {fname, lname} = this.props;
           return (
            <div>
              <h3> firstName:{fname} </h3>
              <h3> LastName:{lname} </h3>
            </div>
Component default Child;
```

Functional components

How to create a function component

2 ways:

1. arrow function
2. function

create a HelloWorld function component

1. using arrow function

```
const HelloWorld = () => <h1>Hello World</h1>;
export default HelloWorld;
```

2. using function

```
function HelloWorld() {
    return <h1>HelloWorld</h1>;
}

export default HelloWorld;
```

Q) Create a functional component to implement Counter

```
import React, {useState} from 'react';
const CounterDemo = () => {
    const [count, setCount] = useState(0);
    const incHandler = () => {
        if (count < 20)
            setCount(count + 1);
    };
    const decHandler = () => {
        if (count > 1)
            setCount(count - 1);
    };
    return (
        <div>
            <h1> {count} </h1>
            <button onClick = {incHandler}> Increment
                                            </button>
            <button onClick = {decHandler}> Decrement
                                            </button>
        </div>
```

```
    )
}
export default CounterDemo;
```

4/10/24

TodoListFuncDemo.js

```
import React, {useState} from 'react'
const TodoListFuncDemo = () => {
    const [item, setItem] = useState('');
    const [items, setItems] = useState([]);
    const addItem = () => {
        setItems( [...items, item]);
        setItem('');
    };
    const deleteItem = (index) => {
        setItems(items.filter( (item, id) => id !== index));
    };
    return (
    <div>
    <input type="text" value={item} placeholder=
        "Add Item" onChange = {(e) => setItem(e.target
                                .value)} />
           
        <button onClick = {addItem}> ADD </button>
        <ol>
        { items.map((item, index) => (
            <li key = {index} > {item}
        <button onClick = { () => deleteItem(index)}>
            Delete </button></li>
        )
    )}
    </ol>
    </div>
    )
}
export default TodoListFuncDemo;
```

useEffect Hook:-

Syntax: useEffect(function, dependency);

componentDidMount:

useEffect(function, []);
↓
callback function

componentDidUpdate:
useEffect(function, [state/props]);

componentWillUnmount:
return () => {

}

15/10

Context API

Steps to work with context API in class based component

1. creating Global context

```
React.createContext();
```

UserContext.jsx

```
import React from 'react'
export const UserContext = React.createContext();
```

2. Invoking Provider Component

Parent.jsx

```
import {UserContext} from './UserContext'
const user = {name: 'Mohan', branch: 'CSE'};

<UserContext.Provider value={user}>
  <Child/>
</UserContext.Provider>
```

3. Consuming the value provided by the provider.

Child.jsx

```
import {UserContext} from './UserContext'
```

```
<UserContext.Consumer>
{value => <h1>value</h1>}
</UserContext.Consumer>
```

Program:-

UserContext.jsx

```
import {createContext} from 'react';
export const UserContext = createContext();
```

Parent.jsx

```
import React, {Component} from 'react';
import {UserContext} from './UserContext';
import Child from './Child';
class Parent extends Component {
  render() {
    const user = {name: 'Mohan', qualification: 'M.tech'};
    return (
      <UserContext.Provider value={user}>
        <Child/>
      </UserContext.Provider>
    )
  }
}
export default Parent;
```

Child.jsx

```
import React, {Component} from 'react';
import {UserContext} from './UserContext'
class Child extends Component {
  render() {
    return (
      <div>
      <UserContext.Consumer>
      {value => (
        <div>
```

```
        <h1> Name : {value.name} </h1>
        <h1> Qualification : {value.qualification} </h1>
        </div>
      )}
    </UserContext.Consumer>
    </div>
  )
 }
}
export default Child;
```

Fetch API

```
import React, { useState, useEffect } from 'react'
~~const [users, setUsers] = useState([]);~~
const FetchPromiseDemo = () => {
    const [users, setUsers] = useState([]);
    const [loading, setLoading] = usestate(true);
    const [error, setError] = useState(null);
    const url = 'https://jsonplaceholder.typicode.com/users'
    const fetchUsers = () => {
        fetch(url)
        .then(response => {
            if(!response.ok)
                throw new Error("Network Error");
            return response.json();
        })
        .then(data => {
            setUsers(data);
            setLoading(false);
        })
        .catch(error => {
            setError(error);
            setLoading(false);
        })
```

```
};
useEffect(() => {
        fetchUsers();
    }, []);
const UserCollection = users.map((user) => (
    <li key={user.id}>{user.name} - {user.email}
    </li>
));
if (loading)
    return <p>Loading....</p>
if (error)
    return <p>{error.message}</p>
return (
<div>
<h1>List of Users</h1>
<ol>
{UserCollection}
</ol>
</div>
)
};
export default FetchPromiseDemo;
```

AsyncAwait

```
const fetchUsers = async() => {
  try
  {
      const response = await fetch(url);
      if(!response.ok)
          throw new Error("Network Error");
      const data = await response.json();
      setUsers(data);
      setLoading(false);
  }catch(error){
      setError(error);
      setLoading(false);
  }
};
```