Name : T.B.Madhan **Reg No :** 23BCE1087

Web Programming Lab (BCSE203E)
LAB – 12
JavaScript - Canvas, Charts and graphs using plotly.js. and Stack elements using Z-Index

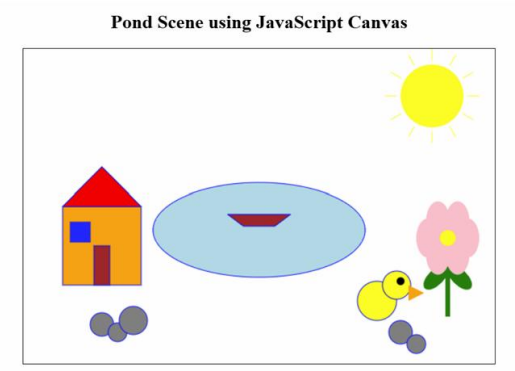
1) **Write a JavaScript program using the HTML5 Canvas API to draw a scene that consists of the following shapes and corresponding drawings:**

Shape	**Drawing Representation**
Oval	Pond
Polygon (Quadrilateral with curved edges)	Boat
Two Circles of Different Sizes	Duck (Body & Head)
A Large Circle with Multiple Straight Lines Extending Outward	Sun
A Rectangle with a Triangle on Top	House
An Ellipse with a Vertical Line and Two Curved Shapes	Flower (Stem, Leaves, and Petals)
Multiple Small Circles	Stones

Requirements:

• Use the Canvas API functions such as arc(), ellipse(), fillRect(), lineTo(), moveTo(), and stroke().
• Assign different colors to each shape.
• Ensure the relative positioning of the elements remains visually structured.

Sample Scene:



2. Apply an animation effect to the boat

Code:

Lab12Q1_2.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
        <title>23BCE1087</title>
        <style>
            canvas {
                border: 1px solid black;
            }
        </style>
    </head>
    <body>
        <canvas id="myCanvas" width="800" height="600"></canvas>

        <script>
            const canvas = document.getElementById("myCanvas");
            const ctx = canvas.getContext("2d");

            //  Pond (Oval)
            ctx.beginPath();
            ctx.ellipse(400, 300, 150, 100, 0, 0, Math.PI * 2);
            ctx.fillStyle = "lightblue";
            ctx.fill();
            ctx.stroke();
```

```
//  Boat
ctx.beginPath();
ctx.moveTo(350, 270);
ctx.lineTo(440, 270);
ctx.lineTo(420, 290);
ctx.lineTo(370, 290);
ctx.closePath();
ctx.fillStyle = "brown";
ctx.fill();
ctx.strokeStyle = "blue";
ctx.stroke();

// Start animation

// Duck
// Duck Body (Ellipse)
ctx.beginPath();
ctx.ellipse(600, 420, 40, 30, 0, 0, Math.PI * 2);
ctx.fillStyle = "yellow";
ctx.fill();
ctx.stroke();

// Duck Head (Circle)
ctx.beginPath();
ctx.arc(630, 400, 20, 0, Math.PI * 2);
ctx.fillStyle = "yellow";
ctx.fill();
ctx.stroke();
//eye
ctx.beginPath();
```

```
            ctx.arc(635, 395, 5, 0, Math.PI * 2);
            ctx.fillStyle = "black";
            ctx.fill();
            ctx.stroke();

            //beak
            ctx.beginPath();
            ctx.moveTo(650, 400);
            ctx.fillStyle = "orange";
            ctx.lineTo(650, 420);
            ctx.lineTo(670, 410);
            ctx.lineTo(650, 400);
            ctx.fill();

            // rays
            for (let i = 0; i < 360; i += 15) {
                let rad = (i * Math.PI) / 180;
                ctx.beginPath();
                ctx.moveTo(650, 100);
                ctx.lineTo(650 + Math.cos(rad) * 80, 100 +
Math.sin(rad) * 80); // Rays
                ctx.strokeStyle = "yellow";
                ctx.stroke();
            }

            //sun
            ctx.beginPath();
            ctx.arc(650, 100, 55, 0, Math.PI * 2);
            ctx.fillStyle = "white";
            ctx.fill();
```

```
ctx.beginPath();
ctx.arc(650, 100, 50, 0, Math.PI * 2);
ctx.fillStyle = "yellow";
ctx.fill();
ctx.stroke();

// House Base (Rectangle)
ctx.fillStyle = "orange";
ctx.fillRect(50, 250, 150, 150);
ctx.strokeStyle = "blue";
ctx.strokeRect(50, 250, 150, 150);

// Roof (Triangle)
ctx.beginPath();
ctx.moveTo(50, 250);
ctx.lineTo(125, 150);
ctx.lineTo(200, 250);
ctx.closePath();
ctx.fillStyle = "red";
ctx.fill();
ctx.stroke();

// Window
ctx.fillStyle = "blue";
ctx.fillRect(65, 275, 40, 40);
ctx.strokeStyle = "Blue";
ctx.strokeRect(65, 275, 40, 40);
// Door
ctx.fillStyle = "Brown";
ctx.fillRect(115, 330, 25, 70);
```

```
ctx.strokeStyle = "Blue";
ctx.strokeRect(115, 330, 25, 70);

// Stones (Multiple small circles)
ctx.beginPath();
ctx.arc(100, 500, 20, 0, Math.PI * 2);
ctx.fillStyle = "gray";
ctx.strokeStyle = "blue";
ctx.lineWidth = 1;
ctx.fill();
ctx.stroke();

ctx.beginPath();
ctx.arc(130, 510, 17, 0, Math.PI * 2);
ctx.fillStyle = "gray";
ctx.fill();
ctx.stroke();

ctx.beginPath();
ctx.arc(150, 490, 23, 0, Math.PI * 2);
ctx.fillStyle = "gray";
ctx.fill();
ctx.stroke();

//Right part
ctx.beginPath();
ctx.arc(650, 490, 20, 0, Math.PI * 2);
ctx.fillStyle = "gray";
ctx.fill();
ctx.stroke();
```

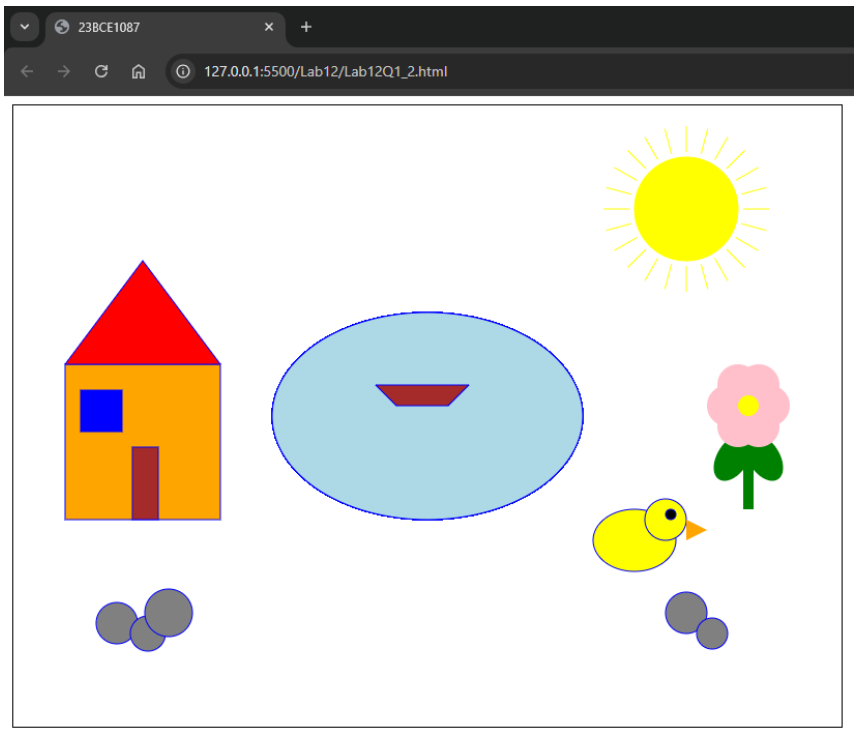
```
ctx.beginPath();
ctx.arc(675, 510, 15, 0, Math.PI * 2);
ctx.fillStyle = "gray";
ctx.fill();
ctx.stroke();

// Stem
ctx.beginPath();
ctx.fillStyle = "green";
ctx.fillRect(705, 310, 10, 80);

//leaf
ctx.ellipse(725, 340, 25, 15, 1, 0, 2 * Math.PI);
ctx.fill();
ctx.closePath();

ctx.beginPath();
ctx.ellipse(695, 340, 25, 15, -1, 0, 2 * Math.PI);
ctx.fill();
ctx.closePath();

//Flowers

ctx.beginPath();
ctx.arc(730, 290, 20, 0, 2 * Math.PI);
ctx.fillStyle = "pink";
ctx.fill();
ctx.closePath();
```

```
ctx.beginPath();
ctx.arc(720, 270, 20, 0, 2 * Math.PI);
ctx.fillStyle = "pink";
ctx.fill();
ctx.closePath();

ctx.beginPath();
ctx.arc(700, 270, 20, 0, 2 * Math.PI);
ctx.fillStyle = "pink";
ctx.fill();
ctx.closePath();

ctx.beginPath();
ctx.arc(690, 290, 20, 0, 2 * Math.PI);
ctx.fillStyle = "pink";
ctx.fill();
ctx.closePath();

ctx.beginPath();
ctx.arc(700, 310, 20, 0, 2 * Math.PI);
ctx.fillStyle = "pink";
ctx.fill();
ctx.closePath();

ctx.beginPath();
ctx.arc(720, 310, 20, 0, 2 * Math.PI);
ctx.fillStyle = "pink";
ctx.fill();
ctx.closePath();
```

```
ctx.beginPath();
ctx.arc(710, 290, 10, 0, 2 * Math.PI);
ctx.fillStyle = "yellow";
ctx.fill();
ctx.closePath();

let boatX = 350; // Initial boat position
let direction = 1; // 1 for right, -1 for left

function drawStaticElements() {
    // Draw Pond (Fixed)
    ctx.beginPath();
    ctx.ellipse(400, 300, 150, 100, 0, 0, Math.PI * 2);
    ctx.fillStyle = "lightblue";
    ctx.fill();
    ctx.stroke();
}

function drawBoat() {
    // Clear only the boat's previous position
    ctx.clearRect(boatX - 5, 260, 100, 40);

    drawStaticElements();
    // Draw Boat (Moving)
    ctx.beginPath();
    ctx.moveTo(boatX, 270);
    ctx.lineTo(boatX + 90, 270);
    ctx.lineTo(boatX + 70, 290);
    ctx.lineTo(boatX + 20, 290);
    ctx.closePath();
```

```
                ctx.fillStyle = "brown";
                ctx.fill();
                ctx.strokeStyle = "blue";
                ctx.stroke();

                // Update boat position
                boatX += direction * 2;
                // Reverse direction if boat reaches pond boundary
                if (boatX > 450 || boatX < 270) {
                    direction *= -1;
                }
                requestAnimationFrame(drawBoat); // Continue
animation
            }
            drawBoat();
        </script>
    </body>
</html>
```

Output:



3) **Write a JavaScript program that creates a working analog clock using the HTML5 Canvas API. The clock should display the current time dynamically and accurately, updating every second.**

Requirements:

i) **Use the Canvas API to draw the clock face, hands, and markings.**
ii) **Ensure the hands move smoothly and update every second.**
iii) **The clock must include the following elements:**
a. A circular clock face with a border and a filled background color.
b. Hour, minute, and second hands that update dynamically based on the current time.
c. Numerical or tick markings for hours (1 to 12).
d. A center pivot point for the hands.

Code:

Lab12Q3.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>23BCE1087</title>
        <style>
            body {
                display: flex;
                justify-content: center;
                align-items: center;
                height: 100vh;
                background-color: #f4f4f4;
            }
            canvas {
                background: white;
                border-radius: 50%;
                box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
```

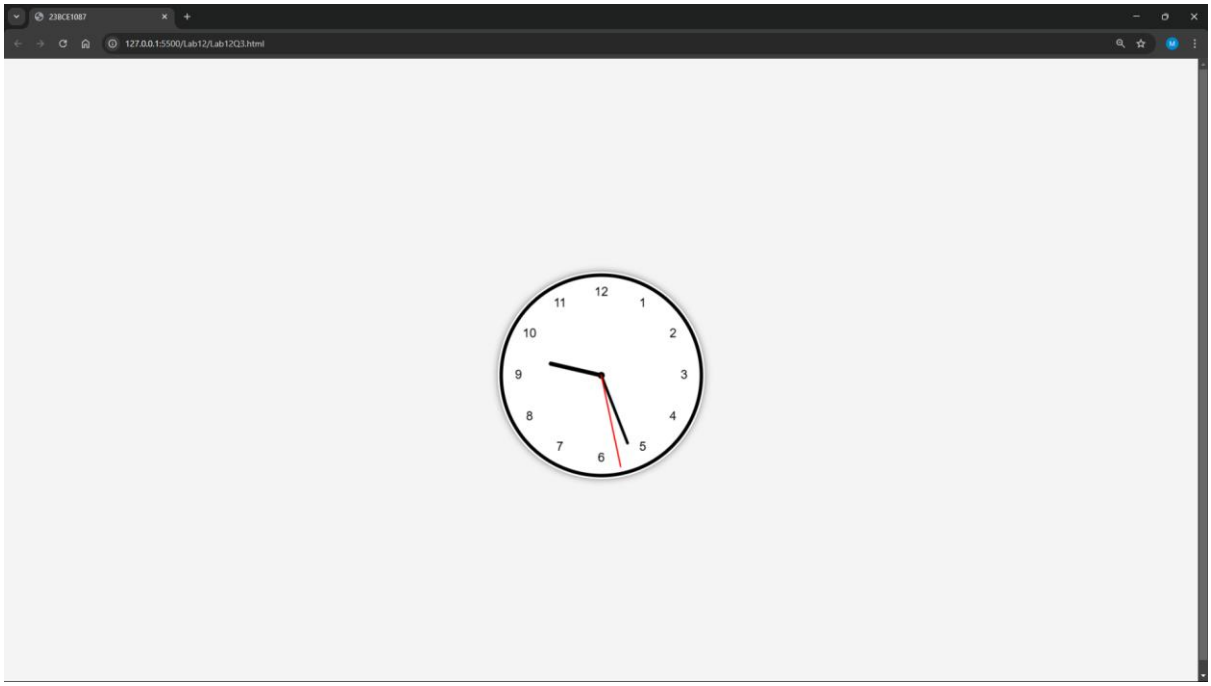
```
        }
    </style>
</head>
<body>
    <canvas id="clockCanvas" width="300" height="300"></canvas>

    <script>
        const canvas = document.getElementById("clockCanvas");
        const ctx = canvas.getContext("2d");
        const radius = canvas.width / 2;

        function drawClock() {
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            ctx.save();
            ctx.translate(radius, radius);

            drawFace(ctx, radius);
            drawNumbers(ctx, radius);
            drawHands(ctx, radius);

            ctx.restore();
        }

        function drawFace(ctx, radius) {
            ctx.beginPath();
            ctx.arc(0, 0, radius - 5, 0, 2 * Math.PI);
            ctx.fillStyle = "white";
            ctx.fill();
            ctx.lineWidth = 5;
            ctx.strokeStyle = "black";
```

```
    ctx.stroke();

    // Center pivot
    ctx.beginPath();
    ctx.arc(0, 0, 5, 0, 2 * Math.PI);
    ctx.fillStyle = "black";
    ctx.fill();
}

function drawNumbers(ctx, radius) {
    ctx.font = "18px Arial";
    ctx.textAlign = "center";
    ctx.textBaseline = "middle";

    for (let num = 1; num <= 12; num++) {
        let angle = (num * Math.PI) / 6;
        let x = (radius - 30) * Math.sin(angle);
        let y = -(radius - 30) * Math.cos(angle);
        ctx.fillText(num, x, y);
    }
}

function drawHands(ctx, radius) {
    const now = new Date();
    const hours = now.getHours() % 12;
    const minutes = now.getMinutes();
    const seconds = now.getSeconds();

    drawHand(
        ctx,
```

```
                    (hours * Math.PI) / 6 + (minutes * Math.PI) /
360,
                    radius * 0.5,
                    6
                );
                drawHand(
                    ctx,
                    (minutes * Math.PI) / 30 + (seconds * Math.PI) /
1800,
                    radius * 0.7,
                    4
                );
                drawHand(ctx, (seconds * Math.PI) / 30, radius *
0.9, 2, "red");
            }

            function drawHand(ctx, angle, length, width, color =
"black") {
                ctx.beginPath();
                ctx.lineWidth = width;
                ctx.lineCap = "round";
                ctx.strokeStyle = color;
                ctx.moveTo(0, 0);
                ctx.rotate(angle);
                ctx.lineTo(0, -length);
                ctx.stroke();
                ctx.rotate(-angle);
            }

            function updateClock() {
                drawClock();
                setTimeout(updateClock, 1000);
```

```
            }

            updateClock();
        </script>
    </body>
</html>
```

Output:



4) **Write a JavaScript program that dynamically generates the charts (bar chart, line chart, pie chart and a donut chart) using Plotly.js.**
   Each chart must include:
   a. Labeled X and Y axes (for bar and line charts).
   b. Title for each chart.
   c. Different colors for data points.
   d. Legend (for the pie chart and donut) showing categories.
   ii) The chart should be scaled properly to fit within the display area.

Code:

Lab12Q4.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>23BCE1087</title>
        <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
        <style>
            body {
                font-family: Arial, sans-serif;
                text-align: center;
                background-color: #f4f4f4;
            }
            .chart-container {
                display: flex;
                flex-wrap: wrap;
                justify-content: center;
                gap: 20px;
                margin-top: 20px;
            }
            .chart {
                width: 45%;
                min-width: 350px;
            }
        </style>
    </head>
    <body>
```

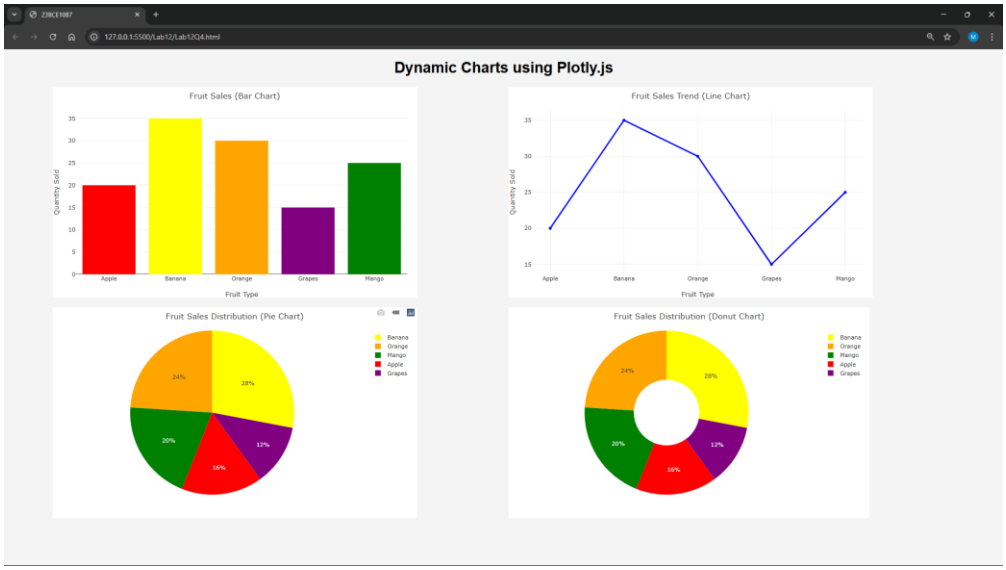
```
        <h1>Dynamic Charts using Plotly.js</h1>
        <div class="chart-container">
            <div id="barChart" class="chart"></div>
            <div id="lineChart" class="chart"></div>
            <div id="pieChart" class="chart"></div>
            <div id="donutChart" class="chart"></div>
        </div>

        <script>
            const categories = ["Apple", "Banana", "Orange",
"Grapes", "Mango"];
            const values = [20, 35, 30, 15, 25];

            // Bar Chart
            Plotly.newPlot(
                "barChart",
                [
                    {
                        x: categories,
                        y: values,
                        type: "bar",
                        marker: {
                            color: [
                                "red",
                                "yellow",
                                "orange",
                                "purple",
                                "green",
                            ],
                        },
                    },
```

```
    ],
    {
        title: "Fruit Sales (Bar Chart)",
        xaxis: { title: "Fruit Type" },
        yaxis: { title: "Quantity Sold" },
        margin: { t: 50, l: 50, r: 20, b: 50 },
    }
);

// Line Chart
Plotly.newPlot(
    "lineChart",
    [
        {
            x: categories,
            y: values,
            type: "scatter",
            mode: "lines+markers",
            line: { color: "blue", width: 3 },
        },
    ],
    {
        title: "Fruit Sales Trend (Line Chart)",
        xaxis: { title: "Fruit Type" },
        yaxis: { title: "Quantity Sold" },
        margin: { t: 50, l: 50, r: 20, b: 50 },
    }
);

// Pie Chart
```

```
Plotly.newPlot(
    "pieChart",
    [
        {
            labels: categories,
            values: values,
            type: "pie",
            marker: {
                colors: [
                    "red",
                    "yellow",
                    "orange",
                    "purple",
                    "green",
                ],
            },
        },
    ],
    {
        title: "Fruit Sales Distribution (Pie Chart)",
        margin: { t: 50, l: 20, r: 20, b: 50 },
    }
);

// Donut Chart (Modified Pie Chart)
Plotly.newPlot(
    "donutChart",
    [
        {
            labels: categories,
```

```
                values: values,
                type: "pie",
                hole: 0.4, // Creates the donut effect
                marker: {
                    colors: [
                        "red",
                        "yellow",
                        "orange",
                        "purple",
                        "green",
                    ],
                },},],
            {
                title: "Fruit Sales Distribution (Donut Chart)",
                margin: { t: 50, l: 20, r: 20, b: 50 },
            }
        );
    </script>
</body>
</html>
```

Output:



5) **Write a JavaScript program that dynamically creates and manipulates overlapping elements using CSS z-index. The program should allow the user to change the stacking order of elements by adjusting their z-index values.**
   • Create at least three overlapping elements (e.g., div boxes or images).
   • Use CSS z-index to control the layering order of these elements.
   • Provide buttons or user input to dynamically adjust the z-index values using JavaScript.
   • Display the current z-index value of each element.

Code:

Lab12Q5.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>23BCE1087</title>
        <style>
            body {
                font-family: Arial, sans-serif;
                text-align: center;
                background-color: #f4f4f4;
            }
            .container {
                position: relative;
                width: 400px;
                height: 400px;
                margin: auto;
                border: 2px solid black;
                overflow: hidden;
            }
            .box {
```

```
        position: absolute;
        width: 150px;
        height: 150px;
        text-align: center;
        line-height: 150px;
        font-weight: bold;
        color: white;
        border: 2px solid black;
        cursor: pointer;
    }
    #box1 {
        background: red;
        top: 50px;
        left: 50px;
    }
    #box2 {
        background: blue;
        top: 100px;
        left: 100px;
    }
    #box3 {
        background: green;
        top: 150px;
        left: 150px;
    }
    .controls {
        margin-top: 20px;
    }
    .controls button {
        margin: 5px;
```

```
            padding: 8px 15px;
            font-size: 14px;
            cursor: pointer;
        }
        .z-index-info {
            margin-top: 10px;
            font-size: 16px;
        }
    </style>
</head>
<body>
    <h1>Dynamic Z-Index Manipulation</h1>

    <div class="container">
        <div id="box1" class="box">Box 1</div>
        <div id="box2" class="box">Box 2</div>
        <div id="box3" class="box">Box 3</div>
    </div>

    <div class="controls">
        <h3>Change Z-Index</h3>
        <label for="boxSelect">Select Box:</label>
        <select id="boxSelect">
            <option value="box1">Box 1</option>
            <option value="box2">Box 2</option>
            <option value="box3">Box 3</option>
        </select>
        <button onclick="moveUp()">Move Up</button>
        <button onclick="moveDown()">Move Down</button>
    </div>
```

```
        <div class="z-index-info">
            <p id="zIndexDisplay"></p>
        </div>

        <script>

            let boxes = document.querySelectorAll(".box");
            let zIndexMap = { box1: 1, box2: 2, box3: 3 };

            function updateZIndexDisplay() {
                document.getElementById("zIndexDisplay").innerHTML =
                    "Box 1: z-index " +
                    zIndexMap["box1"] +
                    "<br>" +
                    "Box 2: z-index " +
                    zIndexMap["box2"] +
                    "<br>" +
                    "Box 3: z-index " +
                    zIndexMap["box3"];
            }

            function moveUp() {
                let selectedBox = document.getElementById(
                    document.getElementById("boxSelect").value
                );
                zIndexMap[selectedBox.id] += 1;
                selectedBox.style.zIndex =
zIndexMap[selectedBox.id];
                updateZIndexDisplay();
            }
```

```
            function moveDown() {
                let selectedBox = document.getElementById(
                    document.getElementById("boxSelect").value
                );
                if (zIndexMap[selectedBox.id] > 1) {
                    zIndexMap[selectedBox.id] -= 1;
                    selectedBox.style.zIndex =
zIndexMap[selectedBox.id];
                    updateZIndexDisplay();
                }
            }
            boxes.forEach((box) => {
                box.style.zIndex = zIndexMap[box.id];
            });
            updateZIndexDisplay();
        </script>
    </body>
</html>
```

Output: