ERODE SENGUNTHAR

ENGINEERING COLLEGE

Autonomous

(Approved by AICTE, New Delhi, Permanently Affiliated to Anna University, Chennai & Accredited by National Board of Accreditation (NBA), New Delhi,
(National Accreditation Assessment Council, Bangalore & IE (I), Kolkata)

PERUNDURAI, ERODE-638057

BONAFIDE CERTIFICATE OF RECORD NOTE

REGISTER NUMBER: ……

Certified that this the Bonafide record of work done By

NAME OF THE STUDENT : _____

BRANCH : _____

SUBJECT CODE : _____

NAME OF THE SUBJECT : _____

FACULTY SIGNATURE **HEAD OF THE DEPARTMENT**

The Record submitted on the end semester practical examination held on **……**

Internal examiner **External Examiner**

TABLE OF CONTENT

EX. NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	MARKS AWARDED	SIGNATURE
11.		Building layouts in Multipoint Perspective			
12.		Object Animation for Water Ripple			
13.		Human Animation and Walk Cycle			
14.		Facial Expressions of Human Animation Cartoons			

EX.NO : 01	**HOW TO WORK WITH PHOTOSHOP**
DATE:	

AIM:

To learn the basic tools, workspace, and operations of Adobe Photoshop and perform basic image editing tasks such as cropping, resizing, color correction, and adding text.

Software Required:

- **Adobe Photoshop (any version – e.g., Photoshop CC, CS6, etc.)**

Procedure:

Step 1: Start Photoshop

- Open **Adobe Photoshop** from the desktop or Start menu.
- Choose **File → New** to create a new project, or **File → Open** to open an existing image.

Step 2: Create a New File

- Go to **File → New**.
- Set:
  - Width and Height (in pixels, cm, or inches)
  - Resolution (72 dpi for web, 300 dpi for print)
  - Background Contents (White or Transparent)
- Click **Create**.

Step 3: Import Image

- Select **File → Open** and browse to the desired image.
- The image appears in a new tab in the workspace.

Step 4: Basic Editing

1. **Crop Image:**
   - Select the **Crop Tool (C)**.
   - Drag to select the area you want to keep.
   - Press **Enter** to apply.
2. **Resize Image:**
   - Go to **Image → Image Size**.
   - Enter new dimensions and click **OK**.
3. **Adjust Brightness/Contrast:**
   - Go to **Image → Adjustments → Brightness/Contrast**.
   - Move sliders and preview changes.

4. **Color Correction:**
    - Use **Image → Adjustments → Hue/Saturation** to change colors.
    - Use **Color Balance** for fine-tuning.

Step 5: Working with Layers

- Open the **Layers Panel (Window → Layers)**.
- Click the **New Layer** icon to add a layer.
- You can rename, hide, duplicate, or delete layers.
- Drag layers up/down to change order.

Step 6: Add Text

- Select **Text Tool (T)**.
- Click on the image and type your text.
- Use the **Options Bar** to change font, size, and color.

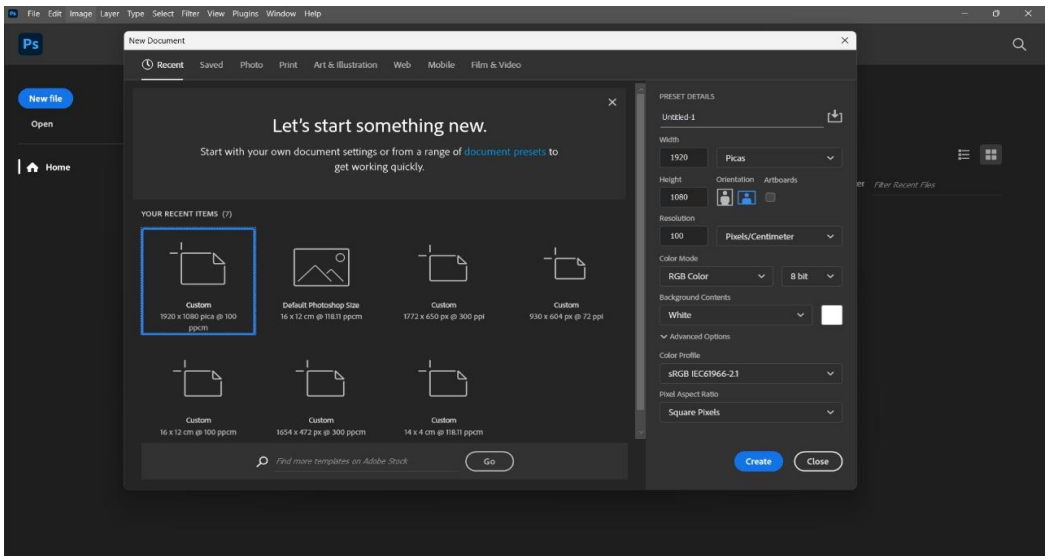
Step 7: Apply Filters

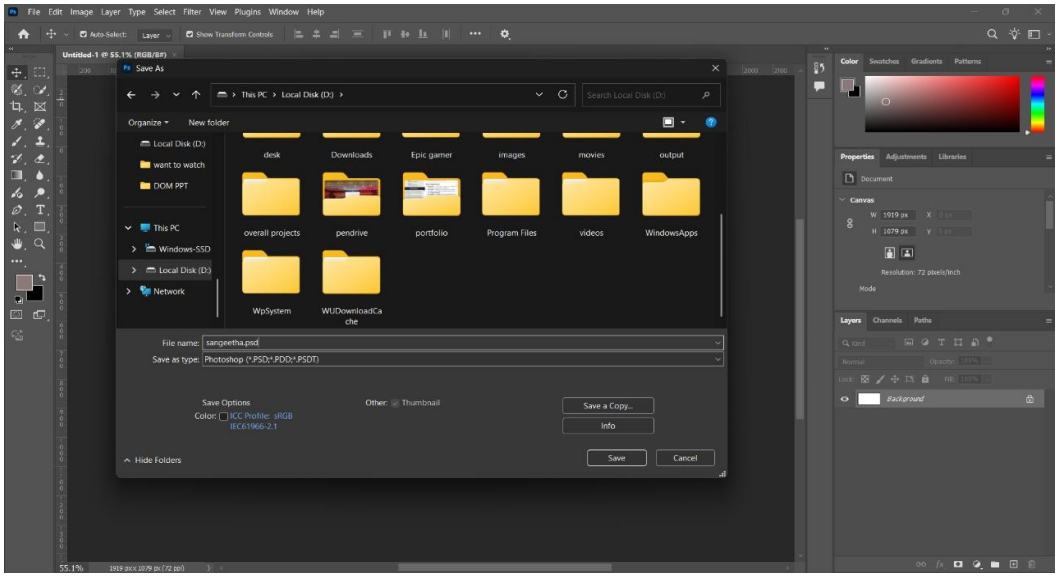
- Go to **Filter → Filter Gallery**.
- Try artistic, blur, or texture filters.
- Adjust settings and preview before applying.

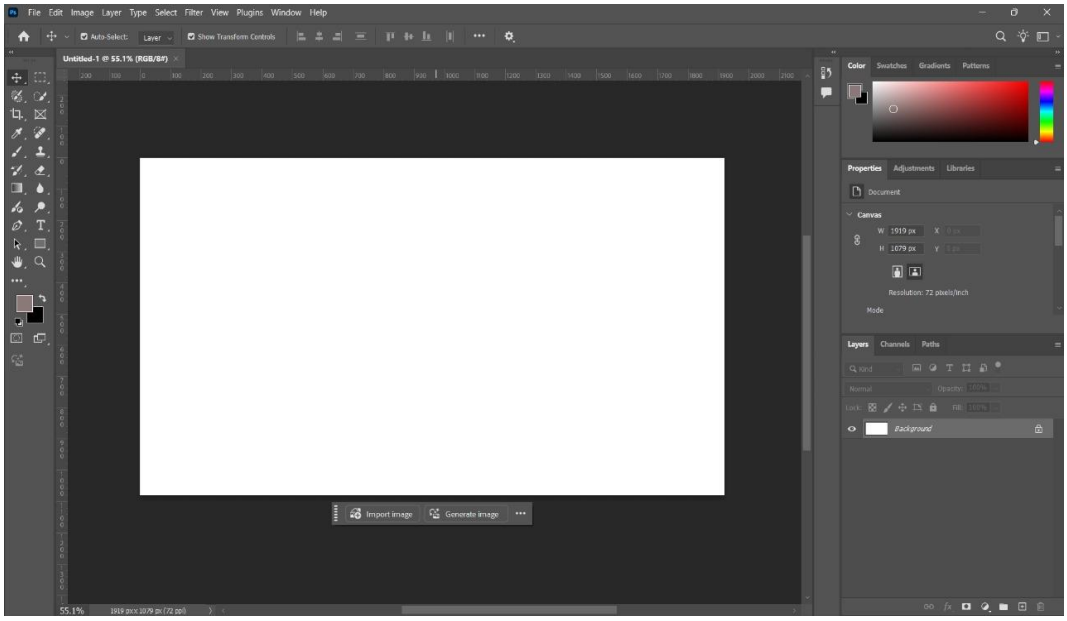
Step 8: Save Your Work

- Save your editable file:
  File → Save As → PSD (Photoshop Document)
  (keeps layers editable)
- Save a final version:
  File → Export → Export As → JPEG / PNG

OUTPUT:







Result:

The image is successfully edited using Photoshop tools and effects.

EX.NO : 02	**WORKING WITH IMAGE EDITING TOOLS – CREATE LOGO AND BANNER USING ADOBE PHOTOSHOP**
DATE:	

AIM:

To learn how to use image editing tools in Adobe Photoshop for creating and designing a Logo and a Banner effectively.

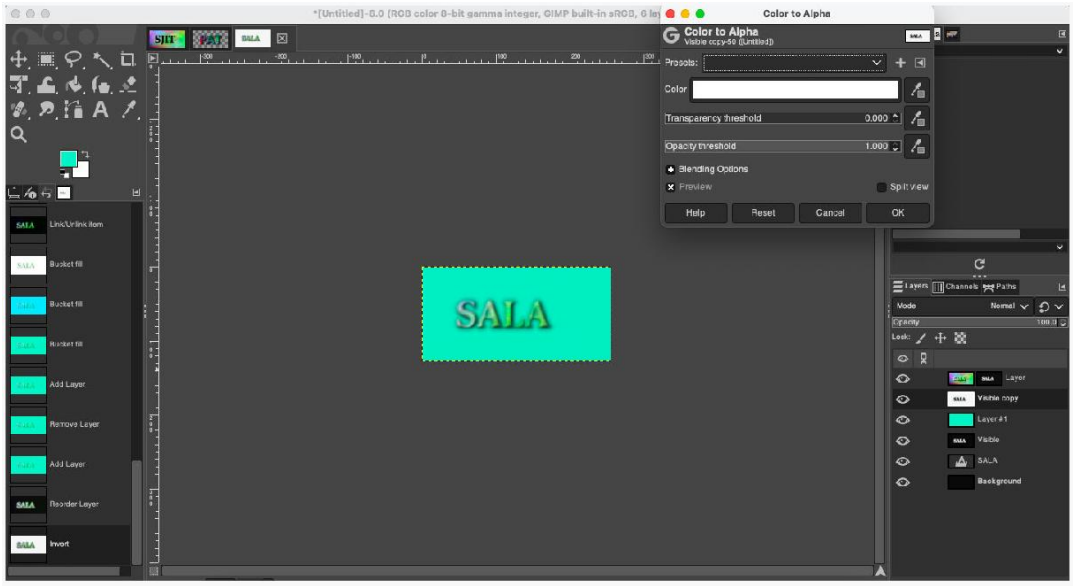
Software Required:

- Adobe Photoshop (CS6/CC or later)

Procedure:

1. **Create a New Document**
   - Go to **File → New**
   - For **Logo:** 1000 × 1000 px (transparent background)
   - For **Banner:** 1200 × 400 px (white or gradient background)
2. **Add Background (for Banner)**
   - Select **Gradient Tool (G)**
   - Choose desired colors and drag to apply gradient background
3. **Design Logo Shape**
   - Use **Shape Tool (U)** to draw basic geometric shapes
   - Use **Pen Tool (P)** for custom shapes
   - Apply **Layer Styles** (Drop Shadow, Bevel & Emboss) for 3D effect
4. **Add Text**
   - Select **Text Tool (T)** and type the brand name or title
   - Adjust **Font**, **Size**, and **Color**
   - Apply **Layer Styles → Gradient Overlay** or **Stroke**
5. **Arrange and Align Elements**
   - Use **Move Tool (V)** to position shapes and text properly
   - Align using top toolbar options or grid guides
6. **Apply Finishing Effects**
   - Add icons, glow, or reflection for creative appeal
   - Adjust overall brightness and contrast for balance
7. **Save and Export**
   - Save editable version as **.PSD**
   - Export logo as **.PNG (transparent)**

OUTPUT:



Result:

Successfully created a professional-looking logo and banner using Photoshop tools.

EX.NO : 03	**WORKING WITH AUDIO EDITING TOOLS**
DATE:	

AIM:

To understand and perform basic audio editing operations such as trimming, mixing, adding effects, and exporting using an audio editing software.

Software Required:

- **Audacity** (Free and open-source audio editor)
  or
- **Adobe Audition / WavePad / Sound Forge** (any available audio editing software)

Procedure:

Step 1: Open the Audio Editing Software

- Launch **Audacity** (or your chosen software).
- Set up the input/output devices under **Edit → Preferences → Devices**.

Step 2: Import or Record Audio

- To record: Click **Record (●)** and speak into the microphone.
- To import: Choose **File → Import → Audio** and select an existing sound file.

Step 3: Select and Trim Audio

- Use the **Selection Tool** to highlight the unwanted parts.
- Press **Delete** to remove them or **Edit → Trim** to keep only the selected region.

Step 4: Adjust Volume

- Use **Effect → Amplify** to increase or decrease loudness.
- Avoid clipping (distortion) by checking preview levels.

Step 5: Apply Noise Reduction

- Select a portion of background noise.
- Go to **Effect → Noise Reduction → Get Noise Profile**.
- Then select the full track → **Effect → Noise Reduction → OK**.

Step 6: Add Effects

- Go to **Effect → Fade In**, **Fade Out**, or **Echo** to add smooth transitions.
- Use **Equalization (EQ)** for tone adjustments.

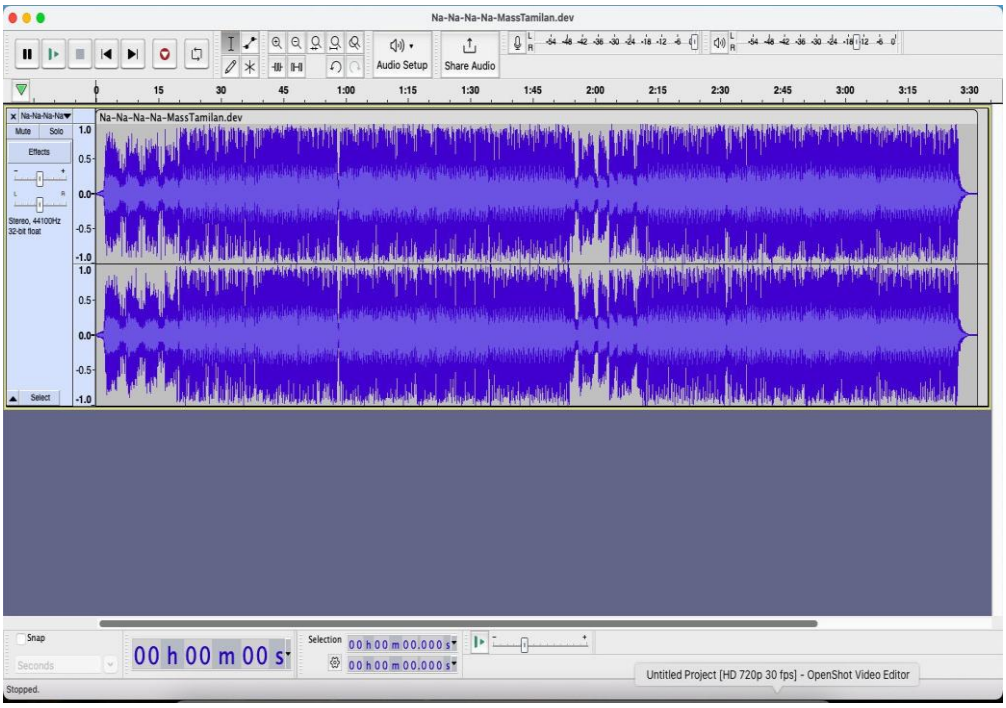
Step 7: Mixing Tracks

- Import multiple audio files (**File → Import → Audio**).
- Use **Time Shift Tool** to align them properly.
- Adjust volume levels for balance.

Step 8: Export the Final Audio

- Save your editable project (**File → Save Project As**).
- Export to desired format:
  File → Export → Export as MP3 / WAV / OGG

OUTPUT:



Result:

Successfully edited, enhanced, and exported an audio file using audio editing tools**.**

EX.NO : 04	**TRANSFORMING SHAPES**
DATE:	

AIM

To create an animation effect using HTML and CSS that transforms a **square** into a **triangle**, and then into a **circle**, demonstrating the use of CSS keyframes and animation properties.

ALGORITHM

Step 1:
Start with an HTML document structure containing a container for the shapes.

Step 2:
Inside the container, create three shapes (square, triangle, circle) using the element.

Step 3:
Use CSS to define the visual styles of each shape:

- The **square** using a normal rectangle with small border-radius.
- The **triangle** using the clip-path: polygon() property.
- The **circle** using border-radius: 50%.

Step 4:
Position all shapes at the same place (using position: absolute) and control their visibility with the opacity property.

Step 5:
Create a **CSS keyframes animation** (@keyframes cycle) that gradually hides one shape and reveals the next (square → triangle → circle → square).

Step 6:
Apply the animation with equal duration and delay for a looping effect.

Step 7:
Save and open the HTML file in a browser to view the animated transformation.

PROGRAM:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Square → Triangle → Circle Morph</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style>
    :root {
      --bg: #0b1020;
      --fg: #e6eefc;
      --accent: #6aa2ff;
    }
    html, body {
      height: 100%;
      margin: 0;
      background: radial-gradient(1200px 800px at 50% 30%, #11193a, var(--bg));
      color: var(--fg);
      font-family: system-ui, -apple-system, Segoe UI, Roboto, sans-serif;
    }
    .wrap {
      min-height: 100%;
      display: grid;
      place-items: center;
      padding: 2rem;
    }
    .card {
```

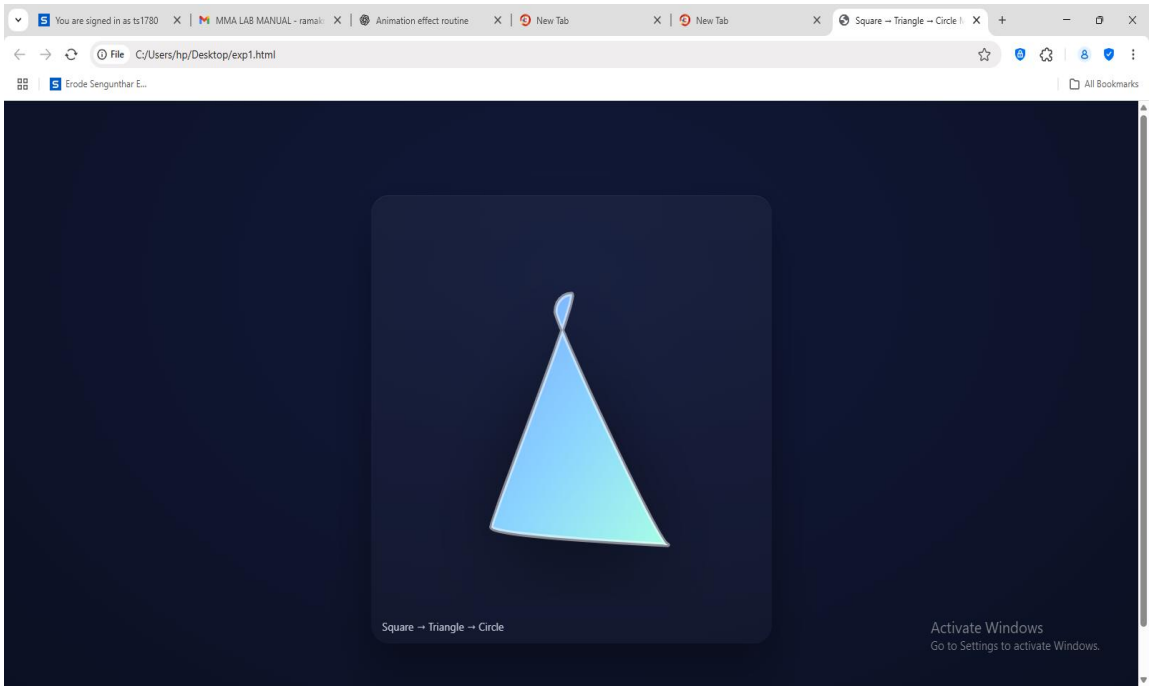
```
  width: min(80vmin, 560px);
  aspect-ratio: 1 / 1;
  border-radius: 24px;
  background: linear-gradient(180deg, rgba(255,255,255,0.04), rgba(255,255,255,0.02));
  box-shadow:
    0 30px 60px rgba(0,0,0,0.35),
    inset 0 1px 0 rgba(255,255,255,0.06);
  display: grid;
  place-items: center;
  position: relative;
  overflow: hidden;
}
.label {
  position: absolute;
  bottom: 12px;
  left: 16px;
  font-size: 14px;
  opacity: 0.8;
  letter-spacing: 0.2px;
}
svg { width: 80%; height: 80%; }
.shape {
  fill: url(#grad);
  stroke: rgba(255,255,255,0.5);
  stroke-width: 2.5;
  filter: drop-shadow(0 10px 24px rgba(0,0,0,0.45));
}
```

```
  </style>
</head>
<body>
  <div class="wrap">
    <div class="card">
      <svg viewBox="0 0 200 200" role="img" aria-label="Shape morph animation">
        <defs>
          <linearGradient id="grad" x1="0" x2="1" y1="0" y2="1">
            <stop offset="0%" stop-color="#7aa8ff"/>
            <stop offset="50%" stop-color="#8ad3ff"/>
            <stop offset="100%" stop-color="#a8ffe6"/>
          </linearGradient>
        </defs>
        <!--
          All three paths below use exactly four cubic curves (C commands),
          so they can morph cleanly between each other.
        -->
        <!-- Initial shape path (square) -->
        <path class="shape"
          d="M40,40
             C40,40 160,40 160,40
             C160,40 160,160 160,160
             C160,160 40,160 40,160
             C40,160 40,40 40,40 Z">
          <!-- Animate the 'd' attribute across the three shapes -->
          <animate attributeName="d"
            dur="6s"
```

```
repeatCount="indefinite"
calcMode="spline"
keyTimes="0;0.33;0.66;1"
keySplines="
  0.4 0 0.2 1;
  0.4 0 0.2 1;
  0.4 0 0.2 1"
values="
  M40,40
  C40,40 160,40 160,40
  C160,40 160,160 160,160
  C160,160 40,160 40,160
  C40,160 40,40 40,40 Z;
  M100,30
  C100,30 30,170 30,170
  C30,170 170,170 170,170
  C170,170 100,30 100,30
  C100,30 100,30 100,30 Z;
  M100,30
  C138.66,30 170,61.34 170,100
  C170,138.66 138.66,170 100,170
  C61.34,170 30,138.66 30,100
  C30,61.34 61.34,30 100,30 Z;
  M40,40
  C40,40 160,40 160,40
  C160,40 160,160 160,160
  C160,160 40,160 40,160
```

```
            C40,160 40,40 40,40 Z
          " />
        </path>
      </svg>
      <div class="label">Square → Triangle → Circle</div>
    </div>
  </div>
</body>
</html>
```

OUTPUT:



RESULT:

Successfully demonstrated the transformation of geometric shapes.

EX.NO : 05	**POOL TABLE SIMULATION**
DATE:	

AIM:

To simulate a pool table where multiple colored balls move and collide with each other using HTML Canvas and JavaScript animation.

ALGORITHM :

1. Create a <canvas> element and set its dimensions and border to resemble a pool table.
2. Define a Ball class with properties like position, velocity, radius, and color.
3. Draw each ball using the arc() function of the Canvas API.
4. Implement physics logic for:
    - Ball movement.
    - Wall collision detection and bouncing.
    - Ball-to-ball collision using distance calculation and velocity swapping.
5. Add friction by reducing the velocity slightly in each animation frame.
6. Use requestAnimationFrame() to continuously update and render the scene for smooth animation.

PROGRAM:

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Pool Table Simulation</title>

<style>

  body {

    background-color: #0b3d0b;

    display: flex;

    justify-content: center;
```

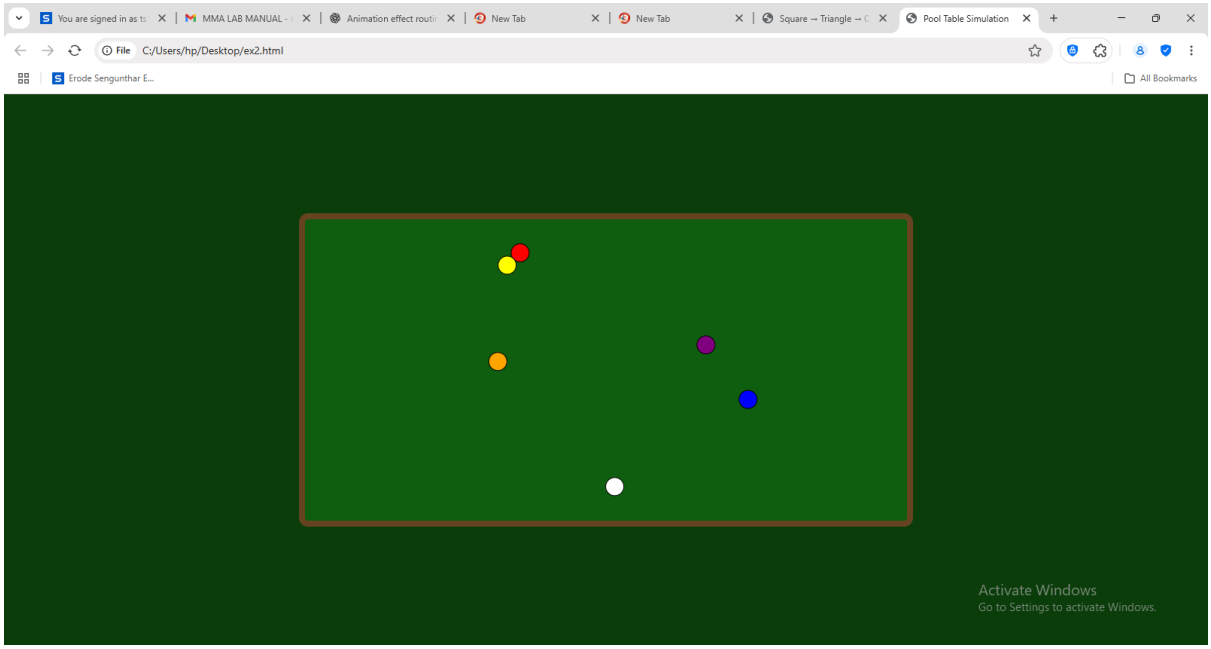
```
    align-items: center;
    height: 100vh;
    margin: 0;
  }
  canvas {
    background-color: #0f5e0f;
    border: 8px solid #654321;
    border-radius: 12px;
  }
</style>
</head>
<body>
<canvas id="poolTable" width="800" height="400"></canvas>
<script>
const canvas = document.getElementById("poolTable");
const ctx = canvas.getContext("2d");
class Ball {
  constructor(x, y, vx, vy, color) {
    this.x = x;
    this.y = y;
    this.vx = vx;
    this.vy = vy;
    this.radius = 12;
    this.color = color;
  }
  draw() {
    ctx.beginPath();
```

```
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2);
    ctx.fillStyle = this.color;
    ctx.fill();
    ctx.strokeStyle = "#000";
    ctx.stroke();
    ctx.closePath();
  }
  update() {
    this.x += this.vx;
    this.y += this.vy;
    // Wall collisions
    if (this.x - this.radius < 0 || this.x + this.radius > canvas.width) {
      this.vx *= -1;
      this.x = Math.max(this.radius, Math.min(this.x, canvas.width - this.radius));
    }
    if (this.y - this.radius < 0 || this.y + this.radius > canvas.height) {
      this.vy *= -1;
      this.y = Math.max(this.radius, Math.min(this.y, canvas.height - this.radius));
    }
    // Friction
    this.vx *= 0.99;
    this.vy *= 0.99;
  }
}
// Create balls
let balls = [];
let colors = ["red", "blue", "yellow", "white", "purple", "orange"];
```

```
for (let i = 0; i < 6; i++) {
  balls.push(new Ball(
    Math.random() * 700 + 50,
    Math.random() * 300 + 50,
    (Math.random() - 0.5) * 6,
    (Math.random() - 0.5) * 6,
    colors[i]
  ));
}
// Ball collision function
function checkCollisions() {
  for (let i = 0; i < balls.length; i++) {
    for (let j = i + 1; j < balls.length; j++) {
      let dx = balls[j].x - balls[i].x;
      let dy = balls[j].y - balls[i].y;
      let dist = Math.sqrt(dx * dx + dy * dy);
      if (dist < balls[i].radius + balls[j].radius) {
        // Simple elastic collision
        let angle = Math.atan2(dy, dx);
        let totalSpeedI = Math.sqrt(balls[i].vx ** 2 + balls[i].vy ** 2);
        let totalSpeedJ = Math.sqrt(balls[j].vx ** 2 + balls[j].vy ** 2);
        // Swap velocities along collision line
        balls[i].vx = totalSpeedJ * Math.cos(angle);
        balls[i].vy = totalSpeedJ * Math.sin(angle);
        balls[j].vx = totalSpeedI * Math.cos(angle + Math.PI);
        balls[j].vy = totalSpeedI * Math.sin(angle + Math.PI);
        // Move apart
```

```
        let overlap = (balls[i].radius + balls[j].radius - dist) / 2;
        balls[i].x -= Math.cos(angle) * overlap;
        balls[i].y -= Math.sin(angle) * overlap;
        balls[j].x += Math.cos(angle) * overlap;
        balls[j].y += Math.sin(angle) * overlap;
      }
    }
  }
}
// Main game loop
function gameLoop() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  checkCollisions();
  balls.forEach(ball => {
    ball.update();
    ball.draw();
  });
  requestAnimationFrame(gameLoop);
}
gameLoop();
</script>
</body>
</html>
```

OUTPUT:



RESULT:

The simulation successfully displayed multiple balls bouncing and colliding realistically inside the canvas area.

EX.NO : 06	**MINESWEEPER GAME**
DATE:	

AIM:

To design and develop an interactive Minesweeper game using HTML, CSS, and JavaScript that allows users to reveal cells, mark flags, and avoid mines.

ALGORITHM :

1. Create a 10×10 grid using HTML elements arranged with CSS Grid.
2. Initialize the game board in JavaScript with cells storing information about:
    - Mine presence
    - Reveal state
    - Flag status
    - Adjacent mine count
3. Randomly distribute a fixed number of mines (e.g., 15) across the grid.
4. Implement left-click functionality to reveal a cell and right-click to place or remove a flag.
5. When a cell is revealed:
    - If it contains a mine → Game Over.
    - If not, display the number of adjacent mines.
    - If zero, recursively reveal neighboring cells.
6. Check win condition when all non-mine cells are revealed.
7. Display appropriate messages (“Game Over” or “You Win”).

PROGRAM:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Minesweeper</title>
<style>
  body {
    background: #1b1b1b;
    color: #fff;
    font-family: Arial, sans-serif;
    text-align: center;
    margin-top: 20px;
  }
  #board {
    display: grid;
    justify-content: center;
    margin: 0 auto;
    border: 3px solid #666;
    background: #333;
  }
  .cell {
    width: 30px;
    height: 30px;
    background: #666;
    border: 1px solid #444;
    display: flex;
```

```
    align-items: center;
    justify-content: center;
    font-weight: bold;
    cursor: pointer;
    user-select: none;
  }
  .cell.revealed {
    background: #ccc;
    color: #000;
    cursor: default;
  }
  .cell.mine {
    background: red;
    color: #fff;
  }
  .cell.flag {
    background: #666;
    color: yellow;
  }
</style>
</head>
<body>
<h1>💣 Minesweeper</h1>
<div id="board"></div>
<p id="status"></p>
<script>
const rows = 10;
```

```
const cols = 10;
const mineCount = 15;
let board = [];
let gameOver = false;
function createBoard() {
  board = [];
  document.getElementById("board").innerHTML = "";
  document.getElementById("board").style.gridTemplateRows = `repeat(${rows}, 30px)`;
  document.getElementById("board").style.gridTemplateColumns = `repeat(${cols}, 30px)`;
  // Initialize board
  for (let r = 0; r < rows; r++) {
    let row = [];
    for (let c = 0; c < cols; c++) {
      let cell = {
        r, c,
        mine: false,
        revealed: false,
        flagged: false,
        adjacentMines: 0,
        element: document.createElement("div")
      };
      cell.element.classList.add("cell");
      cell.element.addEventListener("click", () => revealCell(cell));
      cell.element.addEventListener("contextmenu", (e) => {
        e.preventDefault();
        toggleFlag(cell);
      });
```

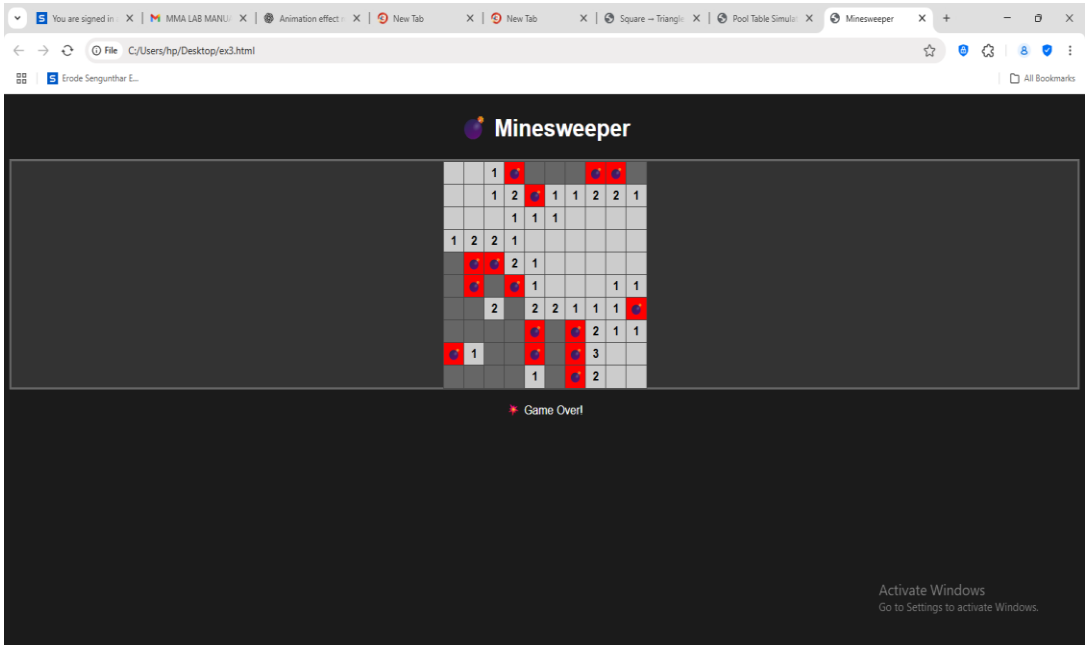
```
      document.getElementById("board").appendChild(cell.element);
      row.push(cell);
    }
    board.push(row);
  }
  // Place mines randomly
  let placed = 0;
  while (placed < mineCount) {
    let r = Math.floor(Math.random() * rows);
    let c = Math.floor(Math.random() * cols);
    if (!board[r][c].mine) {
      board[r][c].mine = true;
      placed++;
    }
  }
  // Calculate numbers
  for (let r = 0; r < rows; r++) {
    for (let c = 0; c < cols; c++) {
      board[r][c].adjacentMines = countAdjacentMines(r, c);
    }
  }
}
function countAdjacentMines(r, c) {
  let count = 0;
  for (let dr = -1; dr <= 1; dr++) {
    for (let dc = -1; dc <= 1; dc++) {
      let nr = r + dr;
```

```
      let nc = c + dc;
      if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && board[nr][nc].mine) {
        count++;
      }
    }
  }
  return count;
}
function revealCell(cell) {
  if (gameOver || cell.revealed || cell.flagged) return;
  cell.revealed = true;
  cell.element.classList.add("revealed");
  if (cell.mine) {
    cell.element.classList.add("mine");
    cell.element.textContent = " 💣 ";
    endGame(false);
    return;
  }
  if (cell.adjacentMines > 0) {
    cell.element.textContent = cell.adjacentMines;
  } else {
    // Reveal neighbors if no adjacent mines
    for (let dr = -1; dr <= 1; dr++) {
      for (let dc = -1; dc <= 1; dc++) {
        let nr = cell.r + dr;
        let nc = cell.c + dc;
        if (nr >= 0 && nr < rows && nc >= 0 && nc < cols) {
```

```
          revealCell(board[nr][nc]);
        }
      }
    }
  }
  checkWin();
}
function toggleFlag(cell) {
  if (gameOver || cell.revealed) return;
  cell.flagged = !cell.flagged;
  cell.element.textContent = cell.flagged ? " ⚑ " : "";
  cell.element.classList.toggle("flag", cell.flagged);
}
function checkWin() {
  let revealedCount = 0;
  for (let r = 0; r < rows; r++) {
    for (let c = 0; c < cols; c++) {
      if (board[r][c].revealed) revealedCount++;
    }
  }
  if (revealedCount === rows * cols - mineCount) {
    endGame(true);
  }
}
function endGame(win) {
  gameOver = true;
  if (!win) {
```

```
    // Reveal all mines
    for (let r = 0; r < rows; r++) {
      for (let c = 0; c < cols; c++) {
        if (board[r][c].mine) {
          board[r][c].element.classList.add("mine");
          board[r][c].element.textContent = " 💣 ";
        }
      }
    }
    document.getElementById("status").textContent = " 💥 Game Over!";
  } else {
    document.getElementById("status").textContent = " 🎉 You Win!";
  }
}
createBoard();
</script>
</body>
</html>
```

OUTPUT:



RESULT:

The interactive Minesweeper game was successfully developed and executed.

EX.NO : 07	**VIDEO EDITING AND CONVERSION TOOLS**
DATE:	

AIM:

To study and perform basic operations using video editing and conversion tools such as trimming, merging, adding effects, and converting video formats (e.g., MP4 to AVI, MOV, or MKV).

PROCEDURE:

1. **Open a Video Editing Tool:**
   Launch any video editing software such as OpenShot, Filmora, VSDC, or Adobe Premiere Pro.

2. **Import the Video File:**
   Load the desired video clip(s) into the editing workspace using the "Import" or "Add Media" option.

3. **Perform Editing Operations:**
   - Trim or cut unwanted parts of the video.
   - Merge multiple clips into a single video.
   - Add text titles, subtitles, transitions, and background music.
   - Apply filters or color correction if needed.

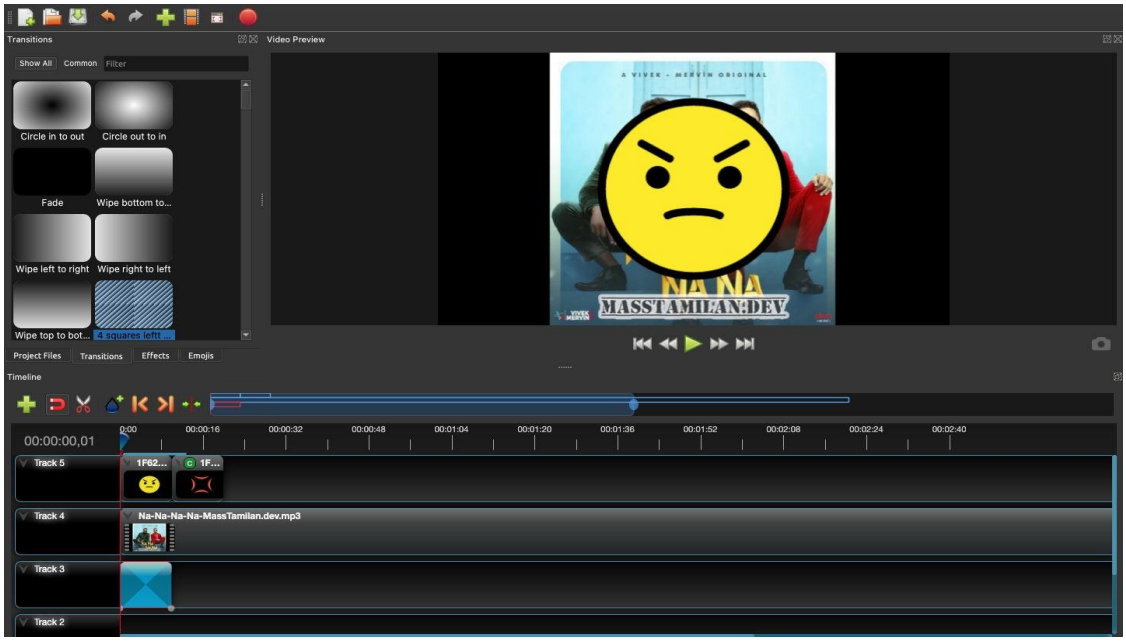
4. **Preview the Edited Video:**
   Play the video in the preview window to verify the applied effects and edits.

5. **Export the Video:**
   Use the "Export" or "Render" option to save the edited video.
   Choose appropriate parameters such as resolution, frame rate, and output format (e.g., MP4, AVI, MOV).

6. **Video Conversion (Format Change):**
   Open a video converter tool (e.g., HandBrake, Format Factory, or Any Video Converter).
   - Import the edited video.
   - Select the target format (e.g., MP4 → AVI).
   - Adjust quality or compression settings if needed.
   - Start the conversion process and save the output file.

7. **Verify Output:**
   Play the converted video to ensure proper playback and quality retention.

OUTPUT:



RESULT:

The video was successfully edited and converted into the desired format using the selected software tools.

EX.NO : 08	**WORKING WITH PAGES AND LAYOUTS IN CorelDRAW**
DATE:	

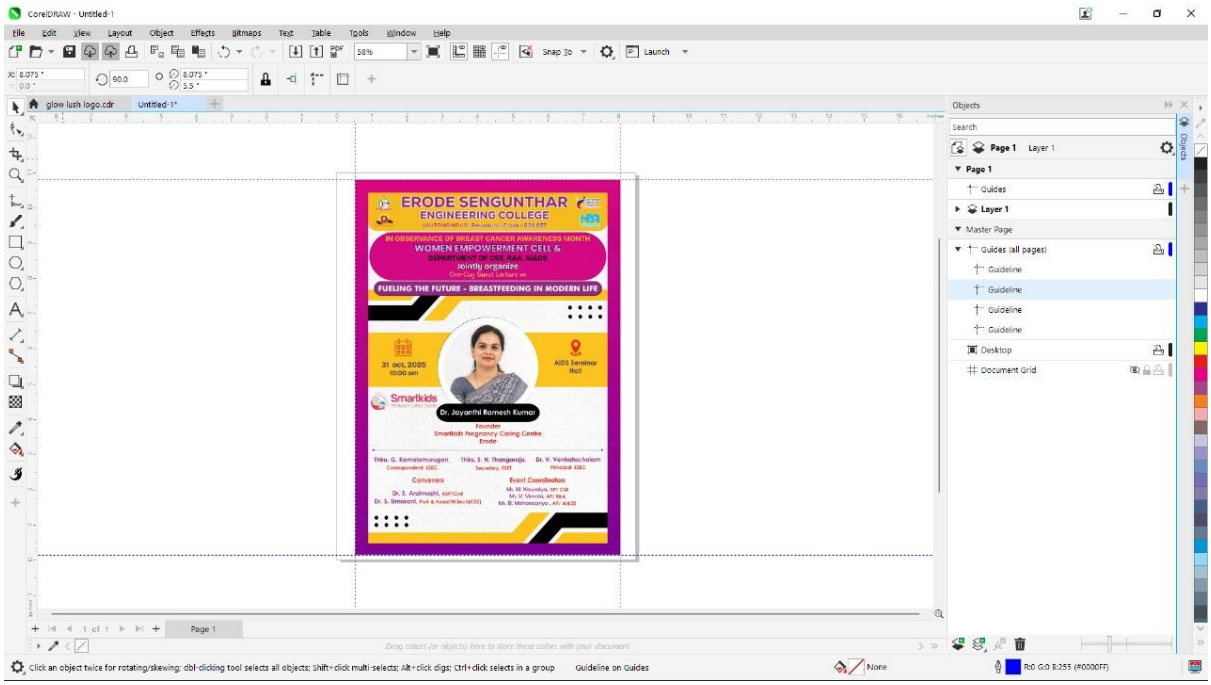
AIM

To learn how to **create, manage, and organize pages and layouts** in **CorelDRAW**, enabling effective design and document structuring for multi-page graphics projects.

PROCEDURE:

1. **Open CorelDRAW:**
   Launch CorelDRAW and create a new document using the *File → New* option.
2. **Set Page Properties:**
   - Define the **page size**, **orientation** (portrait/landscape), and **units** (inches, millimeters, etc.) in the *Page Setup* dialog.
   - Customize **page margins** and **background color** if required.
3. **Add and Manage Pages:**
   - Go to the *Layout → Insert Page* option to add new pages.
   - Use the **Page Navigator** (bottom of the workspace) to switch between pages.
   - Rename, duplicate, or delete pages as needed.
4. **Work with Layouts and Guidelines:**
   - Use the *Layout → Page Setup* menu to adjust layout settings.
   - Add **guidelines**, **gridlines**, and **rulers** to align design objects precisely.
   - Create **master layouts** for headers, footers, or repeated design elements.
5. **Arrange and Align Objects:**
   - Use the *Align and Distribute* tools to position objects uniformly.
   - Apply **page borders**, **columns**, and **bleed settings** for printing layouts.
6. **Preview and Export:**
   - Use *View → Page Sorter View* to preview multiple pages at once.
   - Save and export the project in different formats (CDR, PDF, JPEG, etc.) using the *File → Export* command.

OUTPUT:



RESULT:

The experiment successfully demonstrated how to create and manage multiple pages and layouts in CorelDRAW.

EX.NO : 09	**LOGO DESIGNING IN CorelDRAW**
DATE:	

AIM:

To design a **professional logo** using CorelDRAW by applying basic drawing tools, shapes, colors, text effects, and layout principles.

PROCEDURE:

1. **Open CorelDRAW:**
   Launch CorelDRAW and create a new document using the *File → New* option.
   Set the page size, units, and orientation as required for the logo design.
2. **Plan the Logo Concept:**
   - Decide on the brand name or initials to be represented.
   - Choose suitable shapes, fonts, and colors that align with the brand's identity.
3. **Create Basic Shapes:**
   - Use the **Ellipse Tool**, **Rectangle Tool**, and **Polygon Tool** to draw geometric shapes.
   - Combine or modify shapes using the **Shape Tool**, **Trim**, **Weld**, and **Intersect** options to form unique designs.
4. **Add and Format Text:**
   - Use the **Text Tool (F8)** to insert text for the company or brand name.
   - Choose appropriate **font style**, **size**, and **alignment**.
   - Apply **text effects** such as contour, shadow, or envelope for better styling.
5. **Apply Colors and Fills:**
   - Use the **Smart Fill Tool** or **Color Palette** to add colors.
   - Experiment with **gradient fills**, **transparency**, and **outline thickness** to enhance the appearance.
6. **Arrange and Group Elements:**
   - Align all objects properly using the **Align and Distribute** options.
   - Group related design elements using *Ctrl + G* for easy movement and scaling.

7. **Preview and Export:**
    - Check the logo in *Preview Mode* to ensure clarity and balance.
    - Export the final logo using *File → Export*, selecting a suitable format (PNG, JPEG, or PDF).

OUTPUT:

RESULT:

A creative and well-structured logo was successfully designed using CorelDRAW.

EX.NO : 10	**2D ANIMATION USING BLENDER**
DATE:	

AIM:

To create a **2D animation** using **Blender**, demonstrating keyframing, motion, and object transformations in the 2D workspace.

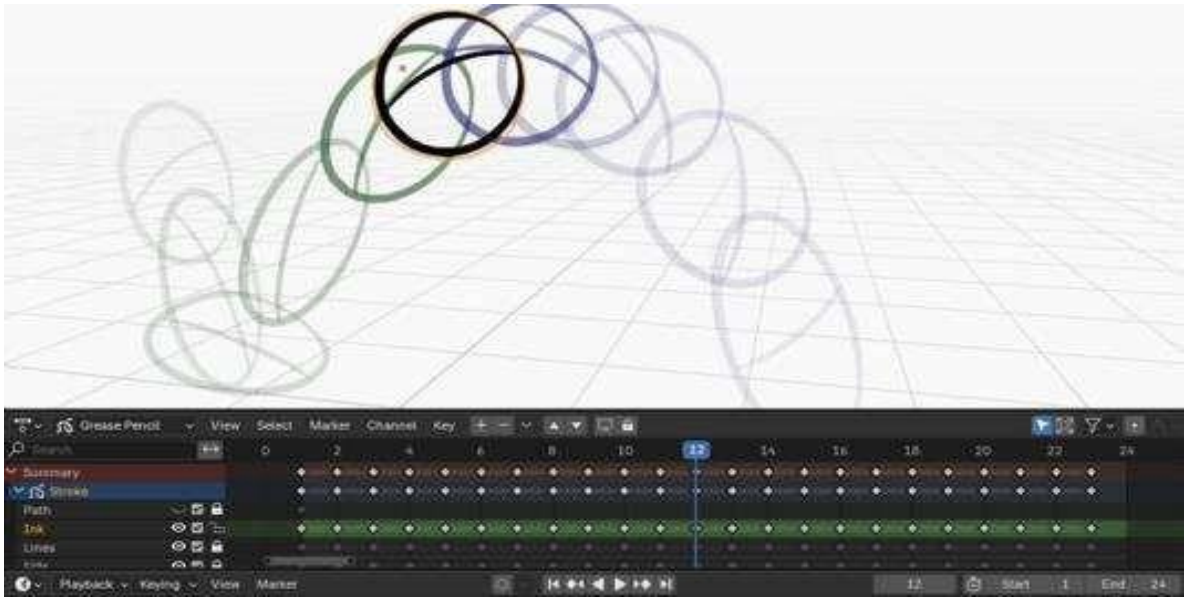
PROCEDURE:

1. **Open Blender:**
    - Launch the Blender application.
    - From the start-up screen, choose **2D Animation** workspace.
2. **Setup the Workspace:**
    - The default layout provides a **Grease Pencil object** for drawing.
    - Switch to the **Draw Mode** to begin sketching on the canvas.
    - Use layers to separate different elements (e.g., background, character, object).
3. **Create the Drawing:**
    - Use the **Grease Pencil Tool** to draw shapes, characters, or scenes.
    - Adjust line thickness, color, and brush type from the **Tool Settings** panel.
4. **Add Keyframes:**
    - Switch to **Animation Mode (Dope Sheet or Timeline)**.
    - Select the frame where movement or change is required.
    - Insert **keyframes** for position, rotation, and scale using the I key.
    - Modify the drawing or object for the next frame to create motion.
5. **Add Motion and Timing:**
    - Adjust the timing between keyframes to control animation speed.
    - Use **Onion Skinning** to preview previous and next frames for smoother transitions.
6. **Preview the Animation:**
    - Play the animation using the **Timeline Controls** or press the **Spacebar**.

- Make adjustments to drawings, timing, or easing curves as needed.

7. **Add Background and Effects:**
   - Create a simple background layer or import an image.
   - Use the **Shader Editor** to apply simple color fills or lighting effects if needed.
8. **Render the Animation:**
   - Open the **Output Properties** tab.
   - Choose output format (MP4, AVI, or Image Sequence).
   - Click **Render → Render Animation** to export the final video.

OUTPUT:



RESULT:

A 2D animation was successfully created and rendered using Blender.

EX.NO : 11	**BUILDING LAYOUTS IN MULTIPOINT PERSPECTIVES**
DATE:	

AIM:

To understand and create **building layouts** using **multipoint perspective drawing techniques**, which help represent realistic architectural forms and spatial depth.

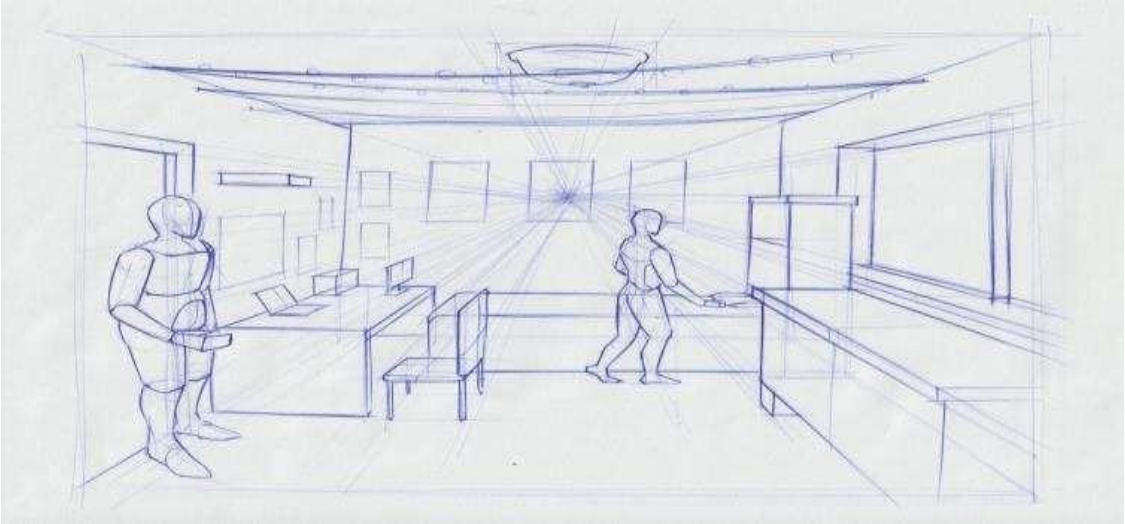
PROCEDURE:

1. **Open Drawing Software / Sketch Setup:**
    - Launch **CorelDRAW**, **Illustrator**, or use **manual drafting tools** (ruler, pencil, and graph sheet).
    - Set up the **workspace** or **canvas** with appropriate dimensions.
2. **Set the Horizon Line and Vanishing Points:**
    - Draw a **horizontal line** across the page — this represents the **eye level** or **horizon line**.
    - Mark **two or three vanishing points** on or near the horizon line depending on the type of perspective:
        - **Two-Point Perspective** – for corner views of buildings.
        - **Three-Point Perspective** – for tall structures viewed from above or below.
3. **Draw the Basic Structure:**
    - Start by sketching the **front edge** or **corner** of the building.
    - Draw **guidelines** from the top and bottom of the edge toward each vanishing point to define depth and height.
4. **Add Building Details:**
    - Use parallel guidelines to outline **walls**, **windows**, **doors**, and **roofs** following the vanishing lines.
    - Repeat the process for adjacent buildings or surrounding structures to create a full layout.
5. **Refine the Perspective:**
    - Darken visible edges and lighten construction lines.

- Add **shadows**, **textures**, and **materials** to enhance realism.
- In digital software, use **layers** to manage different building elements easily.

6. **Add Environment Elements:**
   - Sketch **roads**, **trees**, **skyline**, and **foreground objects** using the same vanishing points for accurate alignment.
   - Apply **color gradients** and **shading** to emphasize light direction.
7. **Finalize the Layout:**
   - Review the alignment and proportions.
   - Save or export the completed design as a project file or image (e.g., JPG, PNG, PDF).

OUTPUT:



RESULT:

A realistic building layout was successfully created using multipoint perspective drawing.

EX.NO : 12	OBJECT ANIMATION FOR WATER RIPPLE
DATE:	

AIM:

To create an **object animation** that simulates **water ripple effects** using animation software such as **Blender**, **Adobe After Effects**, or **Animate CC**.

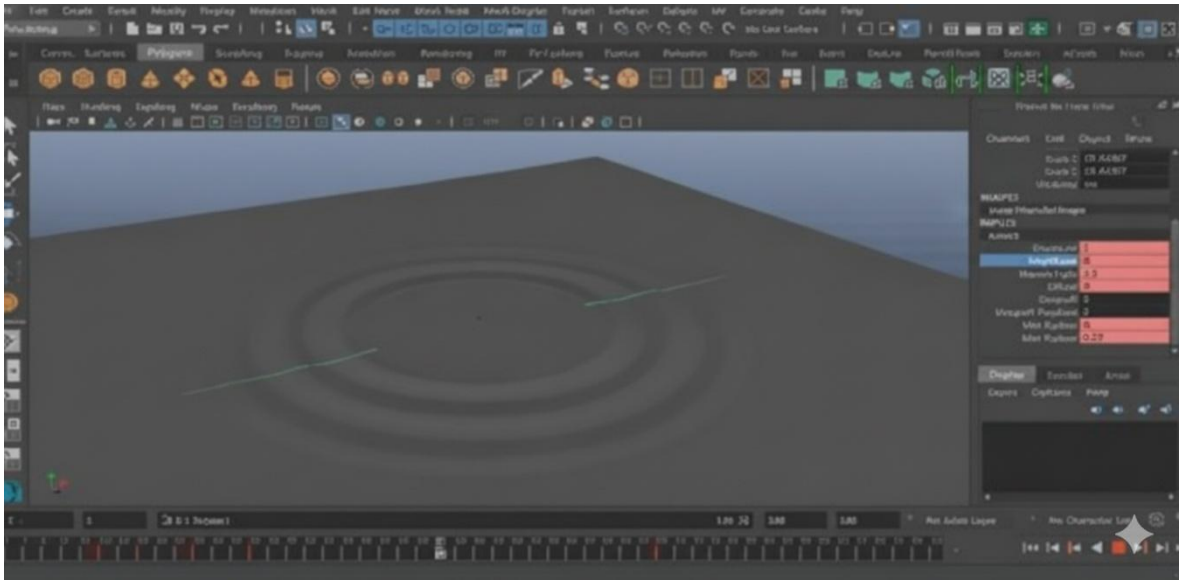
PROCEDURE:

1. **Open Animation Software:**
    - Launch **Blender** (or any preferred animation tool).
    - Create a **new project** and switch to the **Animation** workspace.
2. **Create the Base Object:**
    - Add a **Plane** object to represent the **water surface**.
    - Scale it appropriately using the S key (in Blender) to form a wide surface.
3. **Apply Material and Color:**
    - Open the **Shader Editor** and assign a **blue translucent material** to the plane to represent water.
    - Adjust **roughness** and **transparency** to give a realistic look.
4. **Add Ripple Effect:**
    - Use one of the following methods:
        - **Method 1:** Add a **Wave Modifier** to the plane.
            - Adjust properties such as **Height**, **Width**, **Speed**, and **Damping** to create ripple movement.
        - **Method 2:** Use **Displacement Modifier** with a **cloud or ripple texture** applied to simulate water motion.
        - **Method 3:** Animate **scale and opacity** of circular shapes expanding outward (for 2D animation).
5. **Animate the Ripples:**
    - Insert **keyframes** for wave motion or shape expansion using the timeline.
    - Adjust timing for smooth outward movement and fading effect.

- Enable **looping** for continuous ripple motion.

6. **Add Lighting and Camera:**
   - Insert a **light source** (Sun or Point Light) to create reflections.
   - Adjust **camera angle** to capture the water surface dynamically.
7. **Preview and Render:**
   - Play the animation in the **Timeline** to check the ripple effect.
   - Modify settings for realism if needed (e.g., add multiple ripple sources).
   - Finally, go to **Render → Render Animation** and export it as MP4 or GIF.

OUTPUT:



RESULT:

A water ripple animation was successfully created and rendered.

EX.NO : 13	**HUMAN ANIMATION AND WALK CYCLE**
DATE:	

AIM:

To understand the principles of **human walking motion** and to create a **walk cycle animation** of a character using animation software such as **Blender**, **Maya**, or **Adobe Animate**.

PROCEDURE:

1. **Open the Animation Software:**
    - Launch **Blender** (or any animation software).
    - Create a **new project** and switch to the **Animation workspace**.
2. **Import or Create a Human Model:**
    - Add a **human character model** from the library or design one using modeling tools.
    - Ensure the model has a **rigged skeleton (armature)** for posing.
3. **Setup the Rig:**
    - Parent the model to the armature using **Automatic Weights**.
    - Test the rig by moving joints (legs, arms, head) to ensure smooth bending.
4. **Define the Walk Cycle Stages:**
   A standard **walk cycle** consists of **8 key poses**, representing one complete step:

1. **Contact Pose (Left leg forward)**
2. **Down / Recoil Pose**
3. **Passing Pose**
4. **Up / High Point Pose**
5. **Contact Pose (Right leg forward)**
6. **Down / Recoil Pose**
7. **Passing Pose**
8. **Up / High Point Pose**

- These poses define how weight shifts and how the legs and arms move.

5. **Animate Keyframes:**
   - Move to **Frame 1** in the Timeline and pose the character for the **first contact pose**.
   - Insert a **keyframe** (press I → Location/Rotation).
   - Continue creating each pose in sequence (at frames like 1, 5, 10, 15, etc.).
   - Mirror poses for the opposite leg and arm in the second half of the cycle.
6. **Refine Movement:**
   - Adjust the **arm swing** opposite to the leg motion for natural balance.
   - Fine-tune **timing** and **spacing** using the **Graph Editor**.
   - Add **head tilt**, **torso movement**, and **foot roll** for realistic motion.
7. **Preview the Animation:**
   - Play the animation using the **spacebar** or timeline playback.
   - Adjust frame rate (24–30 FPS) and smooth out transitions.
8. **Lighting and Rendering:**
   - Add a **ground plane**, simple **lighting**, and a **camera** view.
   - Render the final **walk cycle animation** as a video (MP4) or image sequence (PNG).

OUTPUT:

RESULT:

A human walk cycle animation was successfully created and rendered.

EX.NO : 14	FACIAL EXPRESSIONS OF HUMAN ANIMATION CARTOONS
DATE:	

AIM:

To study and create **facial expressions** for human cartoon characters using **animation software** such as **Blender**, **Maya**, or **Adobe Animate**, demonstrating emotions through movement and deformation of facial features.

PROCEDURE:

1. **Open Animation Software:**
    - Launch **Blender** (or any 3D/2D animation tool).
    - Create or import a **cartoon human character model**.
2. **Setup the Character Face:**
    - Ensure the character has a detailed **facial rig** or **control points** for eyes, eyebrows, mouth, and cheeks.
    - If needed, create **shape keys** (in Blender) for facial deformations like smile, frown, blink, etc.
3. **Identify Basic Facial Expressions:**
   The six primary facial expressions to animate are:

1. **Happy** – Eyes open, eyebrows raised, mouth curved upward.
2. **Sad** – Eyes downcast, eyebrows curved upward (middle), mouth corners down.
3. **Angry** – Eyebrows lowered and drawn together, eyes narrowed, lips pressed.
4. **Surprised** – Eyes wide open, eyebrows raised high, mouth open in "O" shape.
5. **Fear** – Eyes wide, eyebrows raised and drawn together, mouth slightly open.
6. **Disgust** – Nose wrinkled, upper lip raised, eyes partly closed.

4. **Create Key Poses:**
    - Use **shape keys (morph targets)** or **control bones** to adjust facial parts for each expression.
    - Insert **keyframes** in the Timeline to record each expression.
    - Gradually blend between neutral and expressive poses for smooth animation.

5. **Animate Eye and Mouth Movements:**
    - Add subtle **eye blinks**, **head tilts**, and **mouth lip-syncing** (if speech is added).
    - Ensure timing matches emotion intensity.
6. **Add Fine Details:**
    - Animate **eyebrow motion**, **cheek puffing**, or **jaw drop** for realism.
    - Refine transitions between expressions using the **Graph Editor**.
7. **Preview and Render:**
    - Play the animation in the Timeline to observe emotional transitions.
    - Adjust lighting and camera angles for clear visibility.
    - Render the final animation in **MP4** or **GIF** format.

OUTPUT:

RESULT:

Facial expressions for a cartoon human character were successfully created and animated.