**HTML5 - Styles and Colors:**

**Canvas - Text and Fonts**

The canvas element allows you to render text and apply different styles and fonts to the text.

The fillText(text, x, y) method is used to draw filled text on the canvas at the specified coordinates.

The strokeText(text, x, y) method is used to draw the outline of the text on the canvas.

You can set the text style properties such as font, text alignment, and color using the font, textAlign, and fillStyle properties of the canvas rendering context.

Example:

const canvas = document.getElementById("myCanvas");

const ctx = canvas.getContext("2d");

ctx.font = "30px Arial";

ctx.textAlign = "center";

ctx.fillStyle = "red";

ctx.fillText("Hello, World!", canvas.width / 2, canvas.height / 2);

ctx.lineWidth = 2;

ctx.strokeStyle = "blue";

ctx.strokeText("Hello, World!", canvas.width / 2, canvas.height / 2 + 50);

**Canvas - Pattern and Shadow:**

The canvas API allows you to use patterns and shadows to enhance the visual appearance of the rendered graphics.

Patterns can be created using images or predefined patterns like gradients.

The createPattern(image, repetition) method is used to create a pattern from an image. The repetition parameter specifies how the pattern should be repeated.

Shadows can be applied to shapes and text using the shadowOffsetX, shadowOffsetY, shadowBlur, and shadowColor properties of the canvas rendering context.

Example:

const canvas = document.getElementById("myCanvas");

const ctx = canvas.getContext("2d");

// Creating a pattern from an image **to FillStyle**

const patternImage = new Image();

patternImage.src = 'C:/Users/Prasanna/Desktop/LevelofMind.JPG';

// Once the image is loaded, create the pattern and draw the shape

patternImage.onload = function() {

// Create the pattern using the image

const **pattern** = ctx.createPattern(patternImage, 'repeat');

// Draw a shape (in this example, a rectangle) using the pattern as the fill style

ctx.beginPath();

ctx.rect(50, 50, 300, 300);

ctx.fillStyle = **pattern**;

ctx.fill();

// Applying a shadow to a rectangle

ctx.shadowOffsetX = 5;

ctx.shadowOffsetY = 5;

ctx.shadowBlur = 10;

ctx.shadowColor = "rgba(0, 0, 0, 0.5)";

ctx.fillStyle = "red";

ctx.fillRect(50, 50, 200, 100);

**Canvas - Save and Restore States**

The canvas API allows you to save and restore the state of the canvas, including transformations, styles, and other attributes.

The save() method is used to save the current state of the canvas, and the restore() method is used to restore the most recently saved state.

This is useful when you want to apply transformations or change styles temporarily and then revert back to the previous state.

Example:

const canvas = document.getElementById("myCanvas");

const ctx = canvas.getContext("2d");

ctx.fillStyle = "red";

ctx.fillRect(50, 50, 100, 100);

ctx.save();

ctx.fillStyle = "blue";

ctx.fillRect(150, 50, 100, 100);

ctx.restore();

ctx.fillRect(250, 50, 100, 100);

**Canvas - Translation, Rotation, Scaling, Transforms**

The canvas API provides methods for translating, rotating, and scaling the coordinate system and the rendered shapes on the canvas.

The translate(x, y) method is used to move the origin of the coordinate system to the specified point.

The rotate(angle) method is used to rotate the coordinate system by the specified angle in radians.

The scale(scaleX, scaleY) method is used to scale the coordinate system and the rendered shapes.

These transformation methods affect all subsequent drawing operations on the canvas.

Example:

const canvas = document.getElementById("myCanvas");

const ctx = canvas.getContext("2d");

// Translation

ctx.translate(100, 100);

ctx.fillRect(0, 0, 100, 100);

// Rotation

ctx.rotate(Math.PI / 4);

ctx.fillRect(0, 0, 100, 100);

// Scaling

ctx.scale(2, 1);

ctx.fillRect(0, 0, 100, 100);

**HTML5 Canvas - Composition**

Composition in HTML5 canvas refers to how multiple shapes and elements are combined to create the final rendered output.

The globalCompositeOperation property of the canvas rendering context determines the composition mode.

Common composition modes include source-over, source-in, source-out, source-atop, destination-over, destination-in, destination-out, destination-atop, lighter, copy, xor, and multiply.

Each composition mode defines how the source image and the destination image are combined.

Example:

const canvas = document.getElementById("myCanvas");

const ctx = canvas.getContext("2d");

// Drawing a red rectangle

ctx.fillStyle = "red";

ctx.fillRect(50, 50, 200, 100);

// Changing the composition mode

ctx.globalCompositeOperation = "destination-over";

// Drawing a blue rectangle

ctx.fillStyle = "blue";

ctx.fillRect(100, 100, 200, 100);

**Canvas – Animations**

The canvas element can be used to create animations by repeatedly redrawing the canvas with different visual states.

Animations can be achieved using techniques like requestAnimationFrame, setInterval, or setTimeout to control the timing of each frame.

Each frame of the animation involves clearing the canvas and drawing the updated scene.

By changing the properties, positions, or transformations of shapes on each frame, you can create dynamic and interactive animations.

Example:

const canvas = document.getElementById("myCanvas");

const ctx = canvas.getContext("2d");

let x = 0;

function drawFrame() {

ctx.clearRect(0, 0, canvas.width, canvas.height);

ctx.fillRect(x, 50, 50, 50);

x += 1;

requestAnimationFrame(drawFrame);

}

drawFrame();

**Create canvas animations**

Canvas animations involve dynamically updating and redrawing the canvas element to create the illusion of motion. This is achieved by repeatedly modifying the canvas content and rendering new frames at a specific interval.

To create canvas animations, you typically follow these steps:

Set up the canvas: Create a canvas element in your HTML document and specify its width and height. You can use the <canvas> tag and set the dimensions using the width and height attributes.

Get the rendering context: Obtain the rendering context of the canvas using JavaScript. The most common rendering context is the 2D context, which can be accessed by calling getContext("2d") on the canvas element.

Define animation variables: Set up variables to control the animation, such as position, speed, direction, or any other properties you want to animate.

Create an animation loop: Use a loop, such as requestAnimationFrame, to repeatedly update and redraw the canvas content. This loop should clear the canvas, modify the animation variables, and render the updated frame.

Update the canvas content: Inside the animation loop, update the values of the animation variables based on the desired animation logic. This could involve changing positions, sizes, colors, or any other properties relevant to your animation.

Draw the updated frame: Using the rendering context, draw the updated frame based on the current state of the animation variables. This could involve drawing shapes, applying transformations, or adding textures and patterns.

Control the animation timing: Use techniques like setTimeout, setInterval, or requestAnimationFrame to control the timing of each frame. This ensures that the animation runs at a desired frame rate and provides smooth motion.

Repeat the animation loop: Keep the animation loop running as long as you want the animation to continue. You can stop the loop when a certain condition is met or when the animation is no longer needed.

By combining these steps and experimenting with different animation techniques and effects, you can create a wide range of dynamic and interactive animations using the HTML5 canvas element.

|  |  |
| --- | --- |
| <!DOCTYPE html>  <html>  <head>  <title>Canvas Animation</title>  <style>  canvas {  border: 1px solid black;  }  </style>  </head>  <body>  <canvas id="myCanvas" width="400" height="200"></canvas>  <script>  const canvas = document.getElementById("myCanvas");  const ctx = canvas.getContext("2d");  let x = 0;  let y = 100;  let dx = 2;  // Movement speed along the x-axis | function drawFrame() {  // Clear the canvas  ctx.clearRect(0, 0, canvas.width, canvas.height);  // Draw a rectangle at the updated position  ctx.fillStyle = "red";  ctx.fillRect(x, y, 50, 50);  // Update the position  x += dx;  // Reverse the movement direction if the rectangle reaches the canvas boundary  if (x + 50 > canvas.width || x < 0) {  dx = -dx;  }  // Call the next frame  requestAnimationFrame(drawFrame);  }  // Start the animation  drawFrame();  </script>  </body>  </html> |