# 9-HTML5 Canvas

# Chapter 9: Exploring the HTML5 Canvas

The HTML5 `<canvas>` element is a powerful tool for rendering graphics directly in the browser. Whether you're creating simple drawings, advanced animations, or interactive games, the canvas is a must-know feature for web developers.

---

## 9.1 What is the HTML5 Canvas?

The `<canvas>` element is like an artist’s canvas—a blank area where you can draw graphics dynamically using JavaScript. It is widely used for:

- Drawing 2D shapes and images.
- Rendering animations.
- Building interactive applications (like games).

**Basic Syntax:**

```
<canvas id="myCanvas" width="500" height="500"></canvas>
```

- The `id` attribute helps identify the canvas in JavaScript.
- The `width` and `height` attributes set the size of the canvas. If omitted, the default size is `300x150` pixels.

---

## 9.2 Setting Up the Canvas

To draw on a canvas, you need to access its **context** using JavaScript. The context provides methods and properties for drawing.

**Example: Getting the Context:**

```
const canvas = document.querySelector("#myCanvas");
const ctx = canvas.getContext("2d"); // 2D rendering context
```

---

## 9.3 Drawing Shapes

The 2D context ( `ctx` ) lets you draw shapes like rectangles, circles, and paths.

1. **Rectangles:**

   - `fillRect(x, y, width, height)` : Draws a filled rectangle.
   - `strokeRect(x, y, width, height)` : Draws the outline of a rectangle.
   - `clearRect(x, y, width, height)` : Clears a rectangular area.

   **Example:**

   ```
   ctx.fillStyle = "blue";
   ctx.fillRect(50, 50, 100, 100); // Filled square
   ```

2. **Lines and Paths:**

   - Use `beginPath()` to start a new path.
   - Use `moveTo(x, y)` to define the starting point.
   - Use `lineTo(x, y)` to draw lines.

   **Example:**

   ```
   ctx.beginPath();
   ctx.moveTo(50, 50);
   ctx.lineTo(150, 50);
   ctx.lineTo(100, 100);
   ctx.closePath();
   ctx.stroke();
   ```

3. **Circles and Arcs:**

   - Use `arc(x, y, radius, startAngle, endAngle)` to draw circles or arcs.

   **Example:**

   ```
   ctx.beginPath();
   ctx.arc(200, 200, 50, 0, Math.PI * 2); // Full circle
   ctx.fillStyle = "red";
   ctx.fill();
   ```

## 9.4 Drawing Text

You can also display text on the canvas.

- `fillText(text, x, y)` : Draws filled text.
- `strokeText(text, x, y)` : Draws outlined text.

**Example:**

```
ctx.font = "20px Arial";
ctx.fillStyle = "black";
ctx.fillText("Hello Canvas!", 50, 50);
```

---

## 9.5 Adding Colors and Styles

Customize the appearance of shapes and lines with the following properties:

1. **Fill Styles:**

   ```
   ctx.fillStyle = "green";
   ctx.fillRect(10, 10, 100, 100);
   ```

2. **Stroke Styles:**

   ```
   ctx.strokeStyle = "blue";
   ctx.lineWidth = 5;
   ctx.strokeRect(10, 10, 100, 100);
   ```

3. **Gradients:**

   ```
   const gradient = ctx.createLinearGradient(0, 0, 200, 0);
   gradient.addColorStop(0, "red");
   gradient.addColorStop(1, "yellow");
   ctx.fillStyle = gradient;
   ctx.fillRect(10, 10, 200, 100);
   ```

---

## 9.6 Handling Images

Images can be drawn on the canvas using the `drawImage()` method.

**Example:**

```
const img = new Image();
img.src = "image.jpg";
img.onload = () => {
  ctx.drawImage(img, 50, 50, 200, 150);
};
```

---

## 9.7 Animating on the Canvas

To create animations, combine the `clearRect()` method with `requestAnimationFrame()` for smooth frame updates.

**Example: A Moving Ball:**

```
let x = 50;
let dx = 2;

function animate() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.beginPath();
  ctx.arc(x, 100, 20, 0, Math.PI * 2);
  ctx.fillStyle = "blue";
  ctx.fill();
  x += dx;

  if (x + 20 > canvas.width || x - 20 < 0) {
    dx = -dx; // Reverse direction
  }

  requestAnimationFrame(animate);
}

animate();
```

---

## 9.8 Handling User Input on the Canvas

Make the canvas interactive by capturing mouse or touch events.

**Example: Drawing with the Mouse:**

```
let drawing = false;

canvas.addEventListener("mousedown", () => (drawing = true));
canvas.addEventListener("mouseup", () => (drawing = false));
canvas.addEventListener("mousemove", (event) => {
  if (!drawing) return;

  ctx.fillStyle = "black";
  ctx.beginPath();
  ctx.arc(event.offsetX, event.offsetY, 5, 0, Math.PI * 2);
  ctx.fill();
});
```

---

## 9.9 Practical Project: Building a Simple Game

**Game: Bouncing Ball**
Create a canvas-based game where a ball bounces around the screen.

**HTML:**

```
<canvas id="gameCanvas" width="600" height="400" style="border: 1px solid black;"></canvas>
```

**JavaScript:**

```
const canvas = document.querySelector("#gameCanvas");
const ctx = canvas.getContext("2d");

let x = canvas.width / 2;
let y = canvas.height / 2;
let dx = 2;
let dy = 2;
const ballRadius = 10;

function drawBall() {
  ctx.beginPath();
  ctx.arc(x, y, ballRadius, 0, Math.PI * 2);
  ctx.fillStyle = "blue";
  ctx.fill();
  ctx.closePath();
}

```

```
function update() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  drawBall();

  if (x + dx > canvas.width - ballRadius || x + dx < ballRadius) {
    dx = -dx; // Reverse horizontal direction
  }

  if (y + dy > canvas.height - ballRadius || y + dy < ballRadius) {
    dy = -dy; // Reverse vertical direction
  }

  x += dx;
  y += dy;

  requestAnimationFrame(update);
}

update();
```

---

## 9.10 Tips for Working with the Canvas

1. **Optimize Performance:**
    - Clear only the areas that change ( `clearRect` ).
    - Use `requestAnimationFrame` for smooth rendering.
2. **Layer Complexity:**
    - Combine multiple canvas elements to separate background, game objects, and effects.
3. **Experiment with Libraries:**
    - Libraries like Konva.js and PixiJS simplify canvas programming.