

Comprehensive Guide to JavaScript and HTML5 Canvas



Learn more HTML, CSS, JavaScript Web Development at <https://basescripts.com/> Laurence Svekis

Welcome to the comprehensive guide on using JavaScript with HTML5 Canvas! This guide will walk you through the fundamentals of the Canvas API, provide detailed code examples and explanations, and include multiple-choice questions and exercises to reinforce your learning.

Comprehensive Guide to JavaScript and HTML5 Canvas	1
1. Introduction to HTML5 Canvas	3
What is HTML5 Canvas?	3
Basic Syntax	3
2. Getting Started with JavaScript and Canvas	4
Accessing the Canvas and Context	4
Exercise 1	5
3. Drawing Shapes	6
Drawing Rectangles	6
Drawing Circles	6
Drawing Lines	7
Exercise 2	7
4. Styling the Canvas	8
Colors, Gradients, and Patterns	8
Solid Colors	8
Gradients	8
Patterns	9
Exercise 3	10
5. Working with Images	10
Drawing Images on Canvas	10
Cropping and Scaling Images	10
Exercise 4	11
6. Creating Animations	12
Using requestAnimationFrame	12
Exercise 5	13
7. Handling User Input	14
Mouse Events on Canvas	14
Keyboard Events	15
Exercise 6	15
8. Advanced Canvas Techniques	17
Working with Paths	17
Drawing Text	17

Compositing and Blending	17
Exercise 7	18
9. Projects and Exercises	19
Project 1: Simple Drawing App	19
Exercise 8	21
10. Multiple Choice Questions	22
Question 1	22
Question 2	23
Question 3	23
Question 4	24
Question 5	24
Question 6	25
Question 7	25
Question 8	25
Question 9	26
Question 10	26
Conclusion	27

1. Introduction to HTML5 Canvas

What is HTML5 Canvas?

The HTML5 `<canvas>` element is a powerful tool for drawing graphics on a web page via scripting (usually JavaScript). It can be used for rendering graphs, game graphics, art, or other visual images.

Basic Syntax

To use the Canvas, include the `<canvas>` element in your HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Canvas Example</title>
```

Learn more HTML, CSS, JavaScript Web Development at <https://basescripts.com/> Laurence Svekis

```
</head>
<body>
  <canvas id="myCanvas" width="500" height="400"
style="border:1px solid #000000;">
    Your browser does not support the HTML5 canvas tag.
  </canvas>
</body>
</html>
```

Explanation:

- id: Unique identifier for the canvas.
 - width and height: Dimensions of the canvas.
 - style: Optional CSS styling, here adding a border for visibility.
-

2. Getting Started with JavaScript and Canvas

To draw on the Canvas, you need to access its drawing context using JavaScript.

Accessing the Canvas and Context

```
<!DOCTYPE html>
<html>
<head>
  <title>Canvas Context</title>
</head>
<body>
  <canvas id="myCanvas" width="500" height="400"></canvas>

  <script>
    // Get the canvas element by ID
    const canvas = document.getElementById('myCanvas');
```

Learn more HTML, CSS, JavaScript Web Development at <https://basescripts.com/> Laurence Svekis

```
// Get the 2D drawing context
const ctx = canvas.getContext('2d');

// Check if context is available
if (ctx) {
    console.log('Canvas context successfully
obtained!');
} else {
    console.error('Canvas not supported in this
browser.');
```

Explanation:

- `getElementById`: Selects the canvas element.
- `getContext('2d')`: Retrieves the 2D rendering context for drawing.

Exercise 1

Task: Modify the above code to change the canvas dimensions to 800x600 and add a light gray background color.

Solution:

```
<canvas id="myCanvas" width="800" height="600"></canvas>

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

    // Fill the canvas with light gray
    ctx.fillStyle = '#D3D3D3';
```

Learn more HTML, CSS, JavaScript Web Development at <https://basescripts.com/> Laurence Svekis

```
    ctx.fillRect(0, 0, canvas.width, canvas.height);  
</script>
```

3. Drawing Shapes

Drawing Rectangles

```
// Draw a filled rectangle  
ctx.fillStyle = '#FF0000'; // Red color  
ctx.fillRect(50, 50, 150, 100); // x, y, width, height  
  
// Draw a rectangle outline  
ctx.strokeStyle = '#0000FF'; // Blue color  
ctx.lineWidth = 5;  
ctx.strokeRect(250, 50, 150, 100);
```

Explanation:

- `fillStyle`: Sets the color to fill the rectangle.
- `fillRect(x, y, width, height)`: Draws a filled rectangle.
- `strokeStyle`: Sets the color for the rectangle's border.
- `lineWidth`: Sets the width of the border line.
- `strokeRect(x, y, width, height)`: Draws a rectangle outline.

Drawing Circles

```
// Begin a new path  
ctx.beginPath();  
  
// Draw a circle  
ctx.arc(150, 300, 50, 0, Math.PI * 2, false);  
  
// Fill the circle
```

```
ctx.fillStyle = 'green';
ctx.fill();

// Outline the circle
ctx.lineWidth = 5;
ctx.strokeStyle = '#003300';
ctx.stroke();
```

Explanation:

- `beginPath()`: Starts a new path for drawing.
- `arc(x, y, radius, startAngle, endAngle, anticlockwise)`: Draws an arc/circle.
- `fill()`: Fills the path with the current fill style.
- `stroke()`: Outlines the path with the current stroke style.

Drawing Lines

```
ctx.beginPath();
ctx.moveTo(400, 50); // Starting point
ctx.lineTo(500, 150); // Ending point
ctx.lineWidth = 2;
ctx.strokeStyle = 'purple';
ctx.stroke();
```

Explanation:

- `moveTo(x, y)`: Moves the pen to a new position without drawing.
- `lineTo(x, y)`: Draws a line from the current position to the new position.

Exercise 2

Task: Draw a blue triangle using the Canvas API.

Solution:

```
ctx.beginPath();
ctx.moveTo(300, 200); // First vertex
ctx.lineTo(350, 300); // Second vertex
ctx.lineTo(250, 300); // Third vertex
ctx.closePath(); // Closes the path back to the first vertex

ctx.fillStyle = 'blue';
ctx.fill();

ctx.strokeStyle = 'black';
ctx.lineWidth = 2;
ctx.stroke();
```

4. Styling the Canvas

Colors, Gradients, and Patterns

Solid Colors

Set the `fillStyle` or `strokeStyle` to a color value.

```
ctx.fillStyle = '#FF69B4'; // Hot pink
ctx.fillRect(100, 400, 200, 100);
```

Gradients

Create linear or radial gradients.

Linear Gradient Example:

```
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);
```

Explanation:

- `createLinearGradient(x0, y0, x1, y1)`: Creates a linear gradient.
- `addColorStop(offset, color)`: Adds colors to the gradient.

Radial Gradient Example:

```
const radGradient = ctx.createRadialGradient(150, 150, 20, 150,
150, 100);
radGradient.addColorStop(0, 'white');
radGradient.addColorStop(1, 'blue');

ctx.fillStyle = radGradient;
ctx.beginPath();
ctx.arc(150, 150, 100, 0, Math.PI * 2, false);
ctx.fill();
```

Patterns

Use images as patterns.

```
const img = new Image();
img.src = 'pattern.png'; // Ensure the image is loaded

img.onload = function() {
  const pattern = ctx.createPattern(img, 'repeat');
  ctx.fillStyle = pattern;
  ctx.fillRect(0, 0, canvas.width, canvas.height);
};
```

Explanation:

- `createPattern(image, repetition)`: Creates a pattern using an image.

Exercise 3

Task: Create a vertical linear gradient from green to blue and apply it to a rectangle.

Solution:

```
const vertGradient = ctx.createLinearGradient(0, 0, 0, 200);
vertGradient.addColorStop(0, 'green');
vertGradient.addColorStop(1, 'blue');

ctx.fillStyle = vertGradient;
ctx.fillRect(400, 200, 150, 200);
```

5. Working with Images

Drawing Images on Canvas

```
const img = new Image();
img.src = 'https://example.com/image.jpg';

img.onload = function() {
  ctx.drawImage(img, 50, 50, 300, 200); // x, y, width, height
};
```

Explanation:

- `drawImage(image, x, y, width, height)`: Draws the image on the canvas.

Cropping and Scaling Images

```
img.onload = function() {
  // Cropping parameters
  const sourceX = 100;
  const sourceY = 50;
  const sourceWidth = 200;
  const sourceHeight = 150;

  // Destination parameters
  const destX = 400;
  const destY = 50;
  const destWidth = 100;
  const destHeight = 75;

  ctx.drawImage(img, sourceX, sourceY, sourceWidth,
sourceHeight, destX, destY, destWidth, destHeight);
};
```

Explanation:

- `drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)`: Draws a cropped and scaled part of the image.

Exercise 4

Task: Load an image and draw it rotated by 45 degrees at the center of the canvas.

Solution:

```
img.onload = function() {
  const centerX = canvas.width / 2;
  const centerY = canvas.height / 2;
  const angle = 45 * Math.PI / 180; // Convert degrees to
radians
```

```
    ctx.save(); // Save the current state
    ctx.translate(centerX, centerY); // Move to center
    ctx.rotate(angle); // Rotate
    ctx.drawImage(img, -img.width / 2, -img.height / 2); // Draw
image centered
    ctx.restore(); // Restore to original state
};
```

6. Creating Animations

Using requestAnimationFrame

`requestAnimationFrame` is a method that tells the browser to perform an animation and requests that the browser calls a specified function to update an animation before the next repaint.

```
let x = 0;
const speed = 2;

function animate() {
    ctx.clearRect(0, 0, canvas.width, canvas.height); // Clear
canvas

    ctx.beginPath();
    ctx.arc(x, canvas.height / 2, 20, 0, Math.PI * 2, false);
    ctx.fillStyle = 'orange';
    ctx.fill();

    x += speed;
    if (x > canvas.width) {
        x = 0;
    }
}
```

```
    requestAnimationFrame(animate);  
}
```

```
// Start the animation  
animate();
```

Explanation:

- `clearRect(x, y, width, height)`: Clears a rectangle area on the canvas.
- The `animate` function updates the position of a circle and redraws it.
- `requestAnimationFrame(animate)`: Recursively calls `animate` for smooth animation.

Exercise 5

Task: Create an animation where a square moves diagonally across the canvas and bounces back when it hits the edges.

Solution:

```
let posX = 50;  
let posY = 50;  
let velocityX = 3;  
let velocityY = 2;  
const squareSize = 50;  
  
function animateSquare() {  
    ctx.clearRect(0, 0, canvas.width, canvas.height);  
  
    // Draw the square  
    ctx.fillStyle = 'cyan';  
    ctx.fillRect(posX, posY, squareSize, squareSize);  
  
    // Update position
```

```
posX += velocityX;
posY += velocityY;

// Check for collision with canvas edges
if (posX + squareSize > canvas.width || posX < 0) {
    velocityX = -velocityX;
}
if (posY + squareSize > canvas.height || posY < 0) {
    velocityY = -velocityY;
}

requestAnimationFrame(animateSquare);
}

animateSquare();
```

7. Handling User Input

Mouse Events on Canvas

You can interact with the canvas by handling mouse events such as `click`, `mousemove`, etc.

```
canvas.addEventListener('click', function(event) {
    const rect = canvas.getBoundingClientRect();
    const x = event.clientX - rect.left;
    const y = event.clientY - rect.top;

    // Draw a small circle where the user clicked
    ctx.beginPath();
    ctx.arc(x, y, 10, 0, Math.PI * 2, false);
    ctx.fillStyle = 'black';
```

Learn more HTML, CSS, JavaScript Web Development at <https://basescripts.com/> Laurence Svekis

```
    ctx.fill();
});
```

Explanation:

- `getBoundingClientRect()`: Gets the size and position of the canvas relative to the viewport.
- Calculates the mouse position relative to the canvas.
- Draws a circle at the clicked position.

Keyboard Events

While Canvas itself doesn't handle keyboard events, you can listen for them on the window or specific elements.

```
window.addEventListener('keydown', function(event) {
    if (event.key === 'ArrowRight') {
        // Move an object to the right
        console.log('Right arrow pressed');
    }
});
```

Exercise 6

Task: Implement dragging functionality for a circle drawn on the canvas.

Solution:

```
let isDragging = false;
let dragCircle = { x: 200, y: 200, radius: 30 };

canvas.addEventListener('mousedown', function(event) {
    const rect = canvas.getBoundingClientRect();
    const mouseX = event.clientX - rect.left;
    const mouseY = event.clientY - rect.top;
```

```
    const distance = Math.hypot(mouseX - dragCircle.x, mouseY -
dragCircle.y);
    if (distance < dragCircle.radius) {
        isDragging = true;
    }
});
```

```
canvas.addEventListener('mousemove', function(event) {
    if (isDragging) {
        const rect = canvas.getBoundingClientRect();
        dragCircle.x = event.clientX - rect.left;
        dragCircle.y = event.clientY - rect.top;
        draw();
    }
});
```

```
canvas.addEventListener('mouseup', function() {
    isDragging = false;
});
```

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

    // Draw the draggable circle
    ctx.beginPath();
    ctx.arc(dragCircle.x, dragCircle.y, dragCircle.radius, 0,
Math.PI * 2, false);
    ctx.fillStyle = 'purple';
    ctx.fill();
}
```

```
draw();
```

8. Advanced Canvas Techniques

Working with Paths

Paths allow for complex shapes by combining lines, arcs, and curves.

```
ctx.beginPath();  
ctx.moveTo(100, 100);  
ctx.lineTo(150, 50);  
ctx.lineTo(200, 100);  
ctx.closePath(); // Closes the path back to the start
```

```
ctx.fillStyle = 'yellow';  
ctx.fill();
```

```
ctx.lineWidth = 2;  
ctx.strokeStyle = 'black';  
ctx.stroke();
```

Drawing Text

You can draw text on the canvas using the `fillText` and `strokeText` methods.

```
ctx.font = '30px Arial';  
ctx.fillStyle = 'black';  
ctx.fillText('Hello, Canvas!', 50, 350);
```

```
ctx.strokeStyle = 'blue';  
ctx.strokeText('Outlined Text', 50, 400);
```

Compositing and Blending

Learn more HTML, CSS, JavaScript Web Development at <https://basescripts.com/> Laurence Svekis

Control how new drawings are combined with existing canvas content.

```
// Set global alpha
ctx.globalAlpha = 0.5;

// Draw two overlapping rectangles
ctx.fillStyle = 'red';
ctx.fillRect(100, 100, 150, 150);

ctx.fillStyle = 'blue';
ctx.fillRect(175, 175, 150, 150);

// Reset global alpha
ctx.globalAlpha = 1.0;
```

Explanation:

- `globalAlpha`: Sets the opacity for all drawing operations.

Exercise 7

Task: Draw a bezier curve and fill it with a gradient.

Solution:

```
const gradient = ctx.createLinearGradient(0, 0, 300, 0);
gradient.addColorStop(0, 'magenta');
gradient.addColorStop(1, 'cyan');

ctx.beginPath();
ctx.moveTo(50, 300);
ctx.bezierCurveTo(150, 100, 250, 500, 350, 300);
ctx.lineWidth = 5;
ctx.strokeStyle = 'black';
ctx.stroke();
```

```
ctx.fillStyle = gradient;
ctx.fill();
```

9. Projects and Exercises

Project 1: Simple Drawing App

Objective: Create a simple drawing application where users can draw with their mouse.

Features:

- Freehand drawing with the mouse.
- Clear canvas button.

Solution:

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple Drawing App</title>
  <style>
    #myCanvas {
      border: 1px solid #000;
      cursor: crosshair;
    }
    #controls {
      margin-top: 10px;
    }
  </style>
</head>
<body>
  <canvas id="myCanvas" width="800" height="600"></canvas>
  <div id="controls">
```

```
    <button id="clearBtn">Clear Canvas</button>
</div>

<script>
  const canvas = document.getElementById('myCanvas');
  const ctx = canvas.getContext('2d');
  let drawing = false;

  canvas.addEventListener('mousedown', function(e) {
    drawing = true;
    ctx.beginPath();
    ctx.moveTo(e.offsetX, e.offsetY);
  });

  canvas.addEventListener('mousemove', function(e) {
    if (drawing) {
      ctx.lineTo(e.offsetX, e.offsetY);
      ctx.strokeStyle = 'black';
      ctx.lineWidth = 2;
      ctx.stroke();
    }
  });

  canvas.addEventListener('mouseup', function() {
    drawing = false;
  });

  canvas.addEventListener('mouseleave', function() {
    drawing = false;
  });
</script>
```

```
document.getElementById('clearBtn').addEventListener('click',
function() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
});
</script>
</body>
</html>
```

Exercise 8

Task: Create a bouncing ball simulation with multiple balls of different sizes and colors.

Solution:

```
const balls = [
  { x: 100, y: 100, dx: 2, dy: 3, radius: 20, color: 'red' },
  { x: 200, y: 150, dx: 3, dy: 2, radius: 30, color: 'green'
},
  { x: 300, y: 200, dx: 4, dy: 1, radius: 25, color: 'blue' }
];
```

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

  balls.forEach(ball => {
    // Draw the ball
    ctx.beginPath();
    ctx.arc(ball.x, ball.y, ball.radius, 0, Math.PI * 2,
false);
    ctx.fillStyle = ball.color;
    ctx.fill();

    // Update position
```

Learn more HTML, CSS, JavaScript Web Development at <https://basescripts.com/> Laurence Svekis

```
    ball.x += ball.dx;
    ball.y += ball.dy;

    // Check for collisions with walls
    if (ball.x + ball.radius > canvas.width || ball.x -
ball.radius < 0) {
        ball.dx = -ball.dx;
    }
    if (ball.y + ball.radius > canvas.height || ball.y -
ball.radius < 0) {
        ball.dy = -ball.dy;
    }
});

    requestAnimationFrame(animateBalls);
}

animateBalls();
```

10. Multiple Choice Questions

Test your understanding with the following multiple-choice questions.

Question 1

What method is used to retrieve the 2D rendering context of a canvas?

- A) `getCanvas('2d')`
- B) `getContext('2d')`
- C) `getRenderingContext('2d')`
- D) `get2DContext()`

Answer: B) `getContext('2d')`

Explanation: The `getContext('2d')` method is used to obtain the 2D rendering context for the canvas.

Question 2

Which of the following is the correct way to draw a filled rectangle at position (50, 50) with width 100 and height 150?

- A) `ctx.drawRect(50, 50, 100, 150);`
- B) `ctx.fillRect(50, 50, 100, 150);`
- C) `ctx.rectangle(50, 50, 100, 150);`
- D) `ctx.drawFilledRect(50, 50, 100, 150);`

Answer: B) `ctx.fillRect(50, 50, 100, 150);`

Explanation: The `fillRect` method draws a filled rectangle with the specified position and dimensions.

Question 3

How can you create a linear gradient that transitions from red to blue horizontally across the canvas?

- A) `ctx.createLinearGradient(0, 0, 0, canvas.height)`
- B) `ctx.createLinearGradient(0, 0, canvas.width, 0)`
- C) `ctx.createLinearGradient(canvas.width, 0, 0, canvas.height)`
- D) `ctx.createLinearGradient(0, 0, canvas.width, canvas.height)`

Answer: B) `ctx.createLinearGradient(0, 0, canvas.width, 0)`

Explanation: To create a horizontal gradient from left to right, the gradient starts at (0,0) and ends at (canvas.width, 0).

Question 4

Which function is commonly used to create smooth animations in the Canvas?

- A) `setInterval()`
- B) `setTimeout()`
- C) `requestAnimationFrame()`
- D) `animate()`

Answer: C) `requestAnimationFrame()`

Explanation: `requestAnimationFrame()` is preferred for creating smooth animations as it is optimized for performance.

Question 5

What does the `beginPath()` method do in Canvas?

- A) It starts a new drawing path.
- B) It closes the current drawing path.
- C) It fills the current path with color.
- D) It clears the canvas.

Answer: A) It starts a new drawing path.

Explanation: `beginPath()` starts a new path by emptying the list of sub-paths.

Question 6

How do you draw an image on the canvas after ensuring it's loaded?

- A) Use `ctx.drawImage()` immediately after setting `src`.
- B) Use `ctx.drawImage()` inside the `onload` event handler of the image.
- C) Use `ctx.drawImage()` before setting `src`.
- D) It is not necessary to wait for the image to load.

Answer: B) Use `ctx.drawImage()` inside the `onload` event handler of the image.

Explanation: To ensure the image is fully loaded before drawing, `drawImage` should be called within the `onload` handler.

Question 7

Which property sets the opacity for all drawing operations on the canvas?

- A) `ctx.opacity`
- B) `ctx.globalOpacity`
- C) `ctx.alpha`
- D) `ctx.globalAlpha`

Answer: D) `ctx.globalAlpha`

Explanation: `globalAlpha` sets the opacity for all subsequent drawing operations.

Question 8

To capture mouse coordinates relative to the canvas, which property of the event object is essential?

- A) `event.pageX` and `event.pageY`
- B) `event.clientX` and `event.clientY`
- C) `event.offsetX` and `event.offsetY`
- D) `event.screenX` and `event.screenY`

Answer: C) `event.offsetX` and `event.offsetY`

Explanation: `offsetX` and `offsetY` provide the mouse coordinates relative to the canvas element.

Question 9

Which method is used to clear a specific area of the canvas?

- A) `ctx.eraseRect(x, y, width, height)`
- B) `ctx.clearCanvas(x, y, width, height)`
- C) `ctx.clearRect(x, y, width, height)`
- D) `ctx.removeRect(x, y, width, height)`

Answer: C) `ctx.clearRect(x, y, width, height)`

Explanation: `clearRect` clears the specified rectangular area, making it fully transparent.

Question 10

What does the `closePath()` method do in Canvas?

- A) It ends the current path without closing it.
- B) It connects the current point to the starting point of the path.
- C) It fills the current path.
- D) It strokes the current path.

Answer: B) It connects the current point to the starting point of the path.

Explanation: `closePath()` creates a straight line from the current point to the starting point of the current path.

Conclusion

This guide provided an overview of the HTML5 Canvas and JavaScript integration, covering basic to advanced techniques. By following the code examples, explanations, and completing the exercises, you should have a solid foundation to create dynamic and interactive graphics on the web. Continue practicing by building projects and exploring the extensive Canvas API to further enhance your skills.