

Hands On: Painting the Canvas

In this lesson, we will discuss the canvas tag in HTML.

WE'LL COVER THE FOLLOWING




- Say hello to `<canvas>`
- **EXERCISE 5-10:** Painting the canvas area
 - Step 1:
 - Step 2:
 - Step 3:
 - Step 4:
 - Step 5:

Today online games infest the web. Without downloading and installing any application, you can directly go to a web page and immediately start an on-line game.

Most of the web pages have amazing graphics and are competitive with many simple games that require local installations. These games are generally implemented with Flash, Silverlight, or other plug-ins that support the hardware acceleration available on your computer.

HTML5 provides a new element, `<canvas>`, which is a place where you can draw. The standard defines a canvas API with operations supporting web applications that demand high performance and low-overhead graphics. You know that you can add JavaScript code to your page, which runs when a web page is loaded, or as you interact with the page. This code can do many things in your browser, in fact it can do everything that the browser's current security configuration allows. If the inline page contains malicious code, that may run on your page.

In this section you will only scratch the surface of the possibilities made available through `<canvas>` and its accompanying API, but it will be enough to understand the concept behind this new HTML5 element.

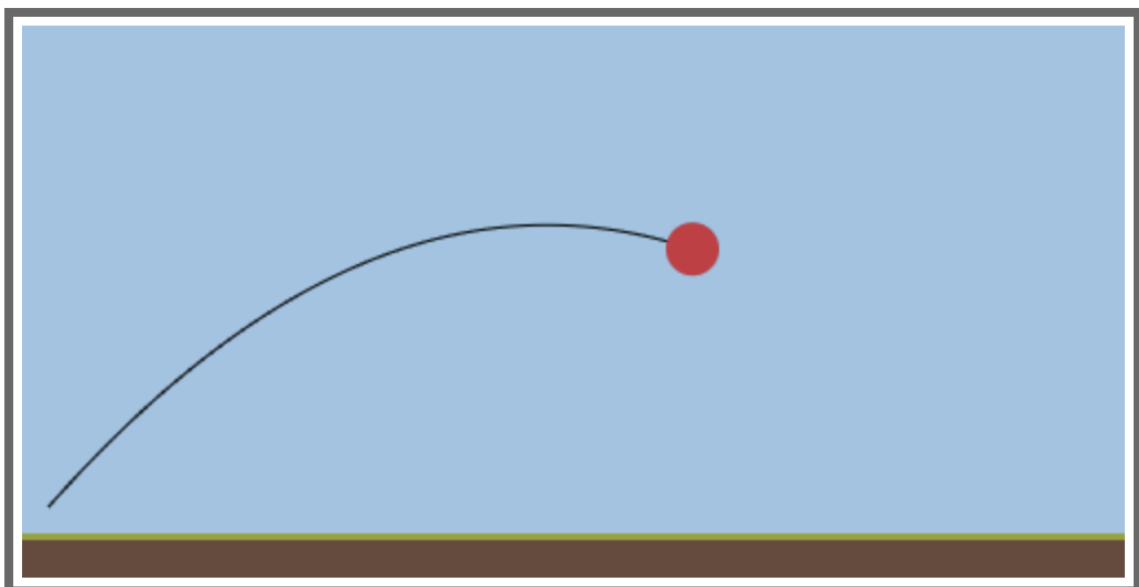
 **NOTE:** To use the canvas API, you must have a basic understanding of JavaScript. Do not feel intimidated if you have not used JavaScript before, the samples in this section are easy to read assuming you have ever used C, C++, C#, VB, or any other imperative programming languages. There are no tricks utilizing any very JavaScript-specific features.

Say hello to `<canvas>`

There is no easier way to understand `<canvas>` than by creating your first page utilizing this element. In the next exercise, you will learn how easy painting the canvas is. You are going to create a simple animation that shows the trajectory of a flung ball.

A momentary snapshot of your final application is shown below:

Painting the Canvas



A snapshot of the final application

The algorithm behind the application is very simple. The trajectory of the ball is calculated from the previous position and the momentary speed of the ball. The ball has an initial speed which is a vector combined from its horizontal and vertical speed. The horizontal speed of the ball is constant; however, its vertical speed is continuously changed by the gravity.

To save time, you are going to start with a prepared Visual Studio solution that can be found in the Exercise-05-10-Begin folder within this chapter's source code download.

EXERCISE 5-10: Painting the canvas area

In this exercise, you are going to paint only a static picture of the animation's scene, including the sky, the soil, the grass, and the ball. To carry out this simple activity, follow these steps:

Step 1:

Open the Exercise-05-10-Begin folder (Exercise-05-10) and use this folder as your starting point.

```
// --- Constants
const HEIGHT = 250;
const WIDTH = 500;
const BALL = 12;
const SOIL = 17;
const GRASS = 3;
const BALLCOLOR = '#CC333F';
const SKYCOLOR = '#9CC4E4';
const SOILCOLOR = '#6A4A3C';
const GRASSCOLOR = '#93A42A';

// --- Drawing context
var ctx;
var ballX;
var ballY;

function initDraw(elemId) {
    ctx = document.getElementById(elemId).getContext('2d');
    ballX = BALL;
    ballY = HEIGHT - BALL - SOIL - GRASS;
    draw();
}

function drawArea() {
    // Drawsky
    ctx.fillStyle = SKYCOLOR;
```

```

ctx.beginPath();
ctx.rect(0, 0, WIDTH, HEIGHT - SOIL - GRASS);
ctx.fill();

// Draw soil
ctx.fillStyle = SOILCOLOR;
ctx.beginPath();
ctx.rect(0, HEIGHT - SOIL, WIDTH, SOIL);
ctx.fill();

// Draw grass
ctx.fillStyle = GRASSCOLOR;
ctx.beginPath();
ctx.rect(0, HEIGHT - SOIL - GRASS, WIDTH, GRASS);
ctx.fill();
}

function draw() {
  drawArea();

  // Draw ball
  ctx.fillStyle = BALLCOLOR;
  ctx.beginPath();
  ctx.arc(ballX, ballY, BALL, 0, Math.PI * 2);
  ctx.closePath();
  ctx.fill();
}

```

Step 2:

In the code editor, open index.html. It contains this markup:

HTML index.html

```

<!DOCTYPE html>
<html>
<head>
  <title>Canvas Tale</title>
  <link href="style.css" rel="stylesheet" />
  <script src="drawing.js"></script>
</head>
<body>
  <h1>Painting the Canvas</h1>
  <!-- Put the canvas here -->
</body>
</html>

```

This file loads the drawing.js script file and links the style sheet to the page. The real “engine” behind the app can be found in drawing.js, which contains a few JavaScript constants, variables, and functions:

JS drawing.js

```

// --- Constants
const HEIGHT = 250;

const WIDTH = 500;
const BALL = 12;
const SOIL = 17;
const GRASS = 3;
const BALLCOLOR = '#CC333F';
const SKYCOLOR = '#9CC4E4';
const SOILCOLOR = '#6A4A3C';
const GRASSCOLOR = '#93A42A';

// --- Drawing context
var ctx;
var ballX;
var ballY;

function initDraw(elemId) {
  ctx = document.getElementById(elemId).getContext('2d');
  ballX = BALL;
  ballY = HEIGHT - BALL - SOIL - GRASS;
  draw();
}

function drawArea() {
  // Draw sky
  ctx.fillStyle = SKYCOLOR;
  ctx.beginPath();
  ctx.rect(0, 0, WIDTH, HEIGHT - SOIL - GRASS);
  ctx.fill();

  // Draw soil
  ctx.fillStyle = SOILCOLOR;
  ctx.beginPath();
  ctx.rect(0, HEIGHT - SOIL, WIDTH, SOIL);
  ctx.fill();

  // Draw grass
  ctx.fillStyle = GRASSCOLOR;
  ctx.beginPath();
  ctx.rect(0, HEIGHT - SOIL - GRASS, WIDTH, GRASS);
  ctx.fill();
}

function draw() {
  drawArea();

  // Draw ball
  ctx.fillStyle = BALLCOLOR;
  ctx.beginPath();
  ctx.arc(ballX, ballY, BALL, 0, Math.PI * 2);
  ctx.closePath();
  ctx.fill();
}

```

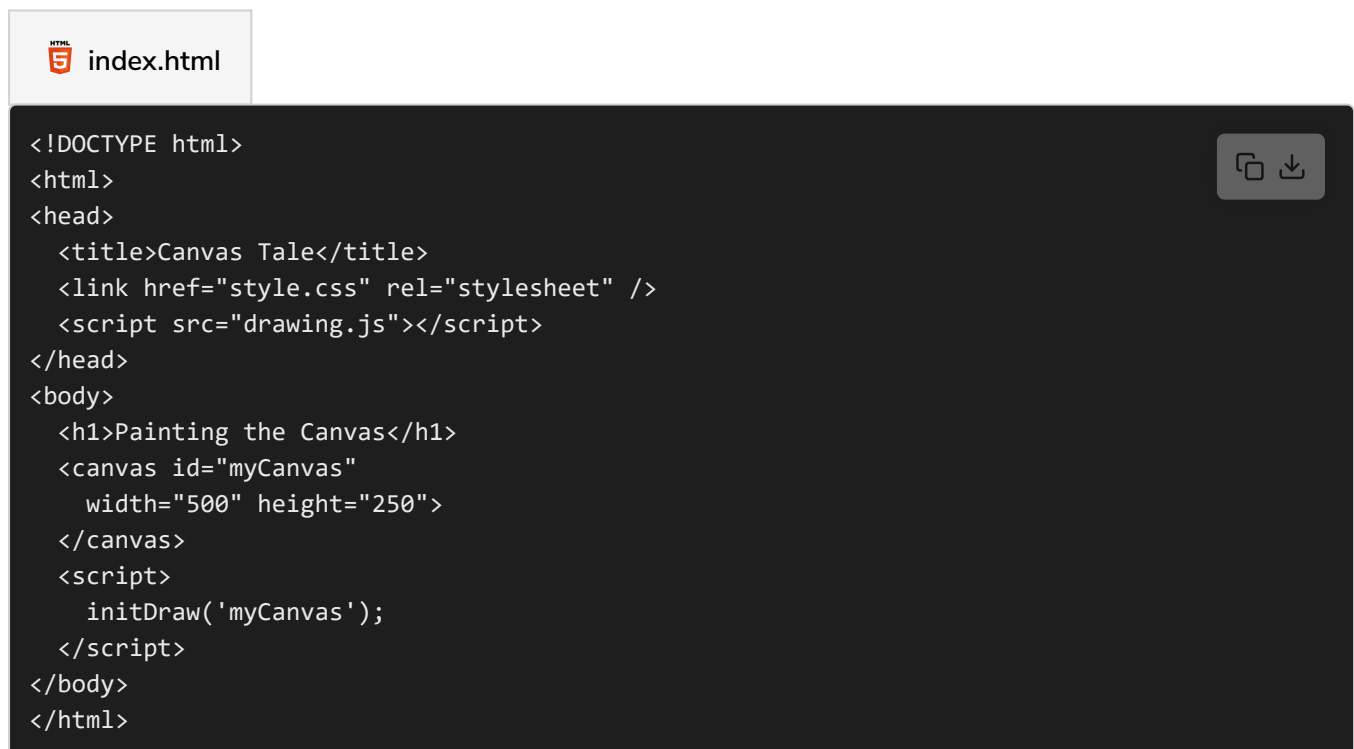
At the end of this exercise, I will explain how this script works. Right now it is enough to know that the `initDraw()` function is responsible to paint the canvas.

Step 3:

Press the run button in the code widget and display the page in your browser via the link provided next to the "Your app can be found at:" message. As you expect, it displays only the "Paint the Canvas" heading, and nothing else.

Step 4:

Switch to the index.html file, and change the placeholder comment to the markup highlighted in this snippet:

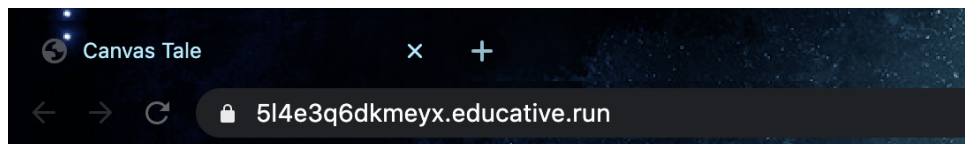


```
<!DOCTYPE html>
<html>
<head>
  <title>Canvas Tale</title>
  <link href="style.css" rel="stylesheet" />
  <script src="drawing.js"></script>
</head>
<body>
  <h1>Painting the Canvas</h1>
  <canvas id="myCanvas"
    width="500" height="250">
  </canvas>
  <script>
    initDraw('myCanvas');
  </script>
</body>
</html>
```

The `<canvas>` tag defines the drawing area. It specifies its identifier as `"myCanvas"` and sets its width and height to 500 and 250 pixels, respectively. The `<canvas>` tag is followed by a `<script>` that invokes the `initDraw()` function passing the identifier of the canvas as its input argument.

Step 5:

Turn back to the browser. Now, it displays the canvas with the scenery already drawn, as shown in the image below:



Painting the Canvas



The scenery is drawn

Great work! In the next lesson, we'll understand the above implementation.

See you there!