**This is the 1st step out of 10 of the**[**Gamedev Canvas tutorial**](https://developer.mozilla.org/en-US/docs/Games/Workflows/Breakout_game_from_scratch)**. You can find the source code as it should look after completing this lesson at**[**Gamedev-Canvas-workshop/lesson1.html**](https://github.com/end3r/Gamedev-Canvas-workshop/blob/gh-pages/lesson01.html)**.**

Before we can start writing the game's functionality, we need to create a basic structure to render the game inside. This can be done using HTML and the [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) element.

**The game's HTML**

The HTML document structure is quite simple, as the game will be rendered entirely on the[<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) element. Using your favourite text editor, create a new HTML document, save it asindex.html, in a sensible location, and add the following code to it:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Gamedev Canvas Workshop</title>

<style>

\* { padding: 0; margin: 0; }

canvas { background: #eee; display: block; margin: 0 auto; }

</style>

</head>

<body>

<canvas id="myCanvas" width="480" height="320"></canvas>

<script>

// JavaScript code goes here

</script>

</body>

</html>

We have a charset defined, [<title>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title) and some basic CSS in the header. The body contains[<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) and [<script>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script) elements — we will render the game inside the first one and write the JavaScript code that controls it in the second one. The [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) element has an id of myCanvasto allow us to easily grab a reference to it, and it is 480 pixels wide and 320 pixels high. All the JavaScript code we will write in this tutorial will go between the opening <script> and closing</script> tags.

**Canvas basics**

To actually be able to render graphics on the [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) element, first we have to grab a reference to it in JavaScript. Add the following below your opening <script> tag.

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

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

Here we're storing a reference to the [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) element to the canvas variable. Then we're creating the ctx variable to store the 2D rendering context — the actual tool we can use to paint on the Canvas.

Let's see an example piece of code that prints a red square on the canvas. Add this below your previous lines of JavaScript, then load your index.html in a browser to try it out.

ctx.beginPath();

ctx.rect(20, 40, 50, 50);

ctx.fillStyle = "#FF0000";

ctx.fill();

ctx.closePath();

All the instructions are between the [beginPath()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/beginPath) and [closePath()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/closePath) methods. We are defining a rectangle using [rect()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/rect): the first two values specify the coordinates of the top left corner of the rectangle on the canvas, while the second two specify the width and height of the rectangle. In our case the rectangle is painted 20 pixels from the left side of the screen and 40 pixels from the top side, and is 50 pixels wide and 50 pixels high, which makes it a perfect square. The[fillStyle](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/fillStyle) property stores a color that will be used by the [fill()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/fill) method to paint the square, in our case, red.

We're not limited to rectangles — here's a piece of code for printing out a green circle. Try adding this to the bottom of your JavaScript, saving and refreshing:

ctx.beginPath();

ctx.arc(240, 160, 20, 0, Math.PI\*2, false);

ctx.fillStyle = "green";

ctx.fill();

ctx.closePath();

As you can see we're using the [beginPath()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/beginPath) and [closePath()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/closePath) methods again. Between them, the most important part of the code above is the [arc()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/arc) method. It takes six parameters:

* x and y coordinates of the arc's center
* arc radius
* start angle and end angle (what angle to start and finish drawing the circle, in radians)
* direction of drawing (false for clockwise, the default, or true for anti-clockwise.) This last parameter is optional.

The [fillStyle](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/fillStyle) property looks different than before. This is because, just as with CSS, color can be specified as a hexadecimal value, a color keyword, the rgba() function, or any of the other available color methods.

Instead of using [fill()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/fill) and filling the shapes with colors, we can use [stroke()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/stroke) to only colour the outer stroke. Try adding this code to your JavaScript too:

ctx.beginPath();

ctx.rect(160, 10, 100, 40);

ctx.strokeStyle = "rgba(0, 0, 255, 0.5)";

ctx.stroke();

ctx.closePath();

The code above prints a blue-stroked empty rectangle. Thanks to the alpha channel in thergba() function, the blue color is semi transparent.

**Compare your code**

Here's the full source code of the first lesson, running live in a JSFiddle:

**Exercise**: try changing the size and color of the given shapes.

**Next steps**