

Computer Science Principles

Web Programming

JavaScript Canvas Programming

CHAPTER 1: STARTER

DR. ERIC CHOU

IEEE SENIOR MEMBER



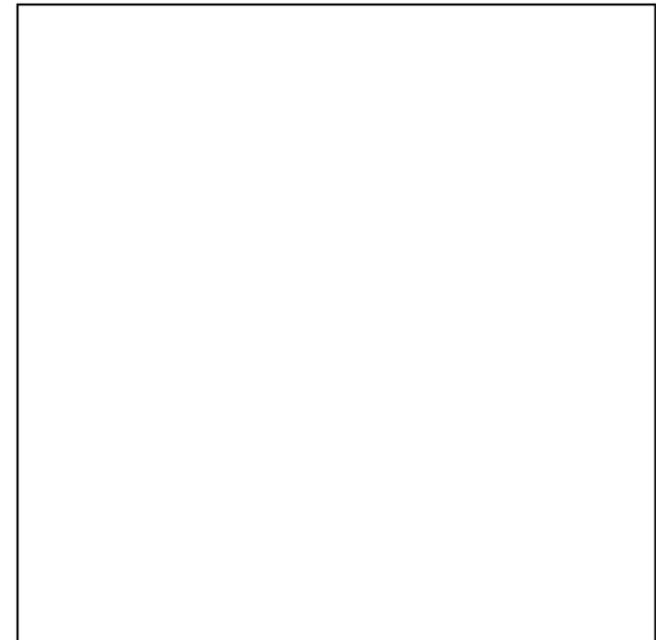
Overview

- Canvas is one of the most sought after feature in HTML5. Developers like to use it for creating rich web applications.
- Users may use those applications without using proprietary browser plug-ins like Adobe's flash player.
- Most of the modern browsers like Chrome, Firefox, Safari, Opera, IE9 and 10 support it. In a moment we will see what canvas is capable of and how you may use it.



What's the Canvas Element For?

- Officially a canvas is "a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly".
- Simply saying, with the help of **JavaScript** and **HTML5 canvas** element you may render 2D shapes and bitmap images. The image below shows the canvas with a black border.



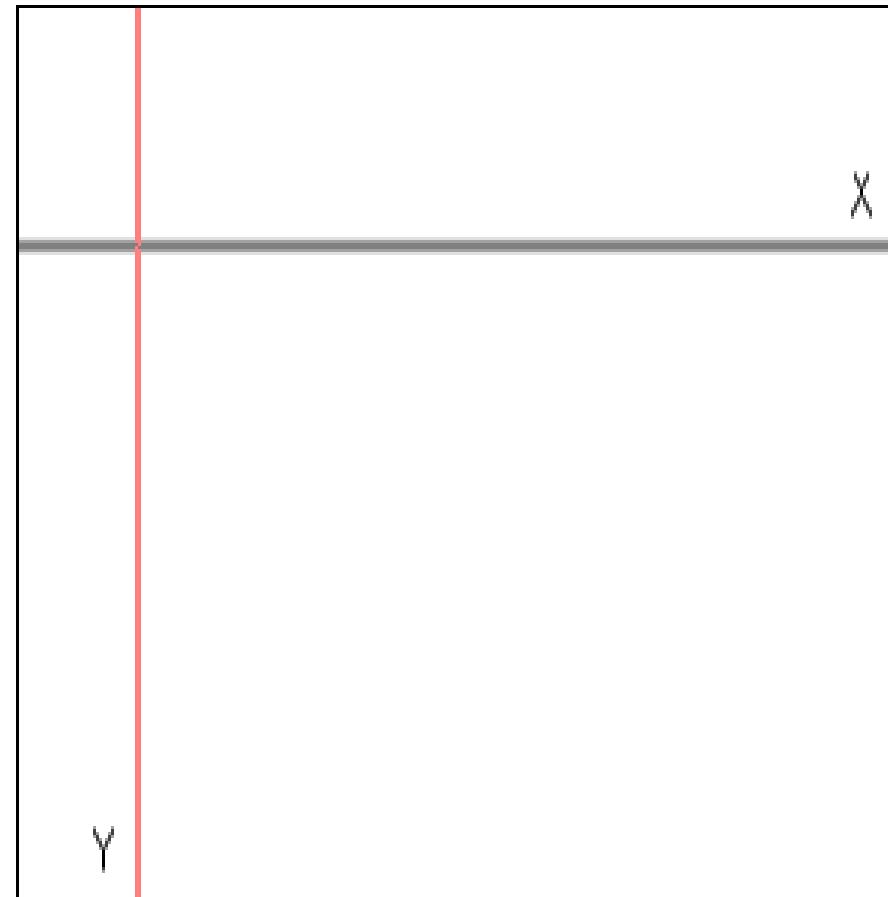


Canvas is an Element Tag and also a painting Canvas

- A webpage may contain multiple canvas elements. Each canvas may have an **id** using which you may target a specific canvas through JavaScript. Each canvas element has a **2D Context**. This again has objects, properties, and methods. Tinkers these, you may draw your stuff. To draw on a canvas, you need to reference the context of the canvas. The context gives you access to the 2D properties and methods that We'll dive deeper into the context later.
- Every canvas element has x and y coordinates. X being the horizontal coordinate and y being the vertical coordinate. The following image shows these coordinates on a canvas.



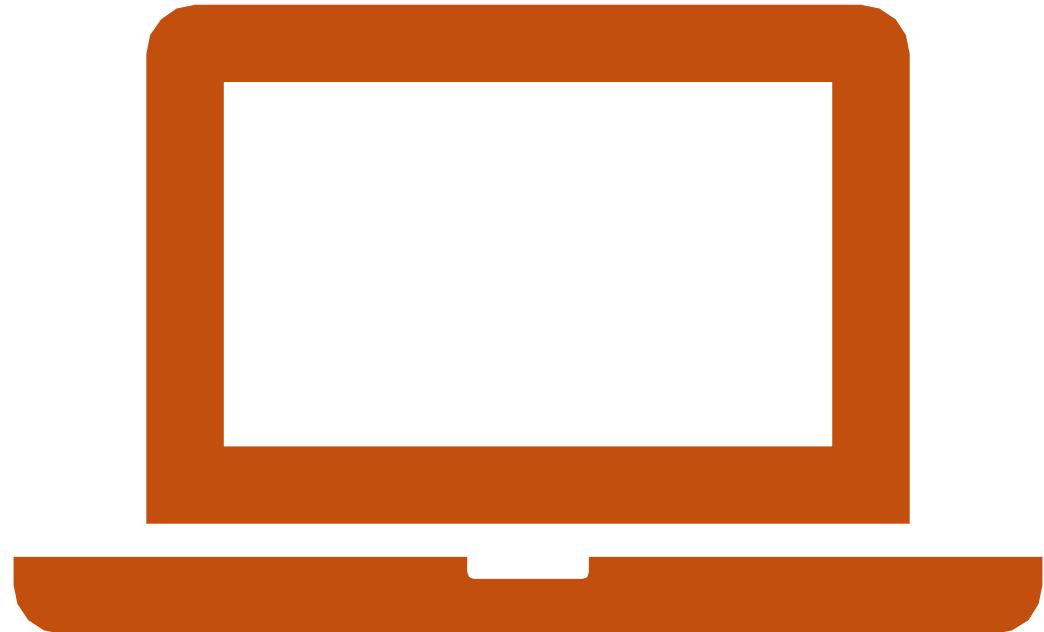
The Coordinate of a Canvas Element





Overview

LECTURE 1



Demonstration Program

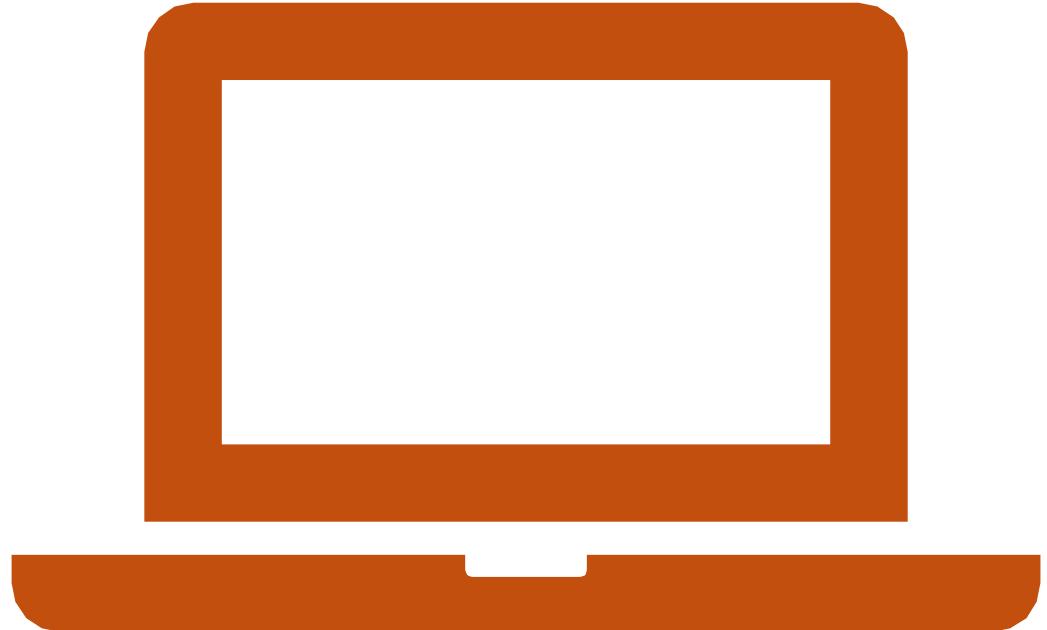
STARTER.HTML

```
1  <!DOCTYPE html>
2  ▼ <html>
3  ▼     <head>
4      <title>HTML5 Canvas Demo</title>
5  ▼     <style>
6  ▼         #FirstCanvas{
7             width: 500px;
8             height: 300px;
9             border: 3px solid green;
10            background-color: orange;
11        }
12    </style>
13  </head>
14 ▼  <body>
15      <canvas id="FirstCanvas"></canvas>
16  </body>
17 </html>
18
```



1. **canvas** is an element tag.
2. canvas is similar to JPanel or Canvas in Java
3. canvas a is a painting area for HTML5 programs.



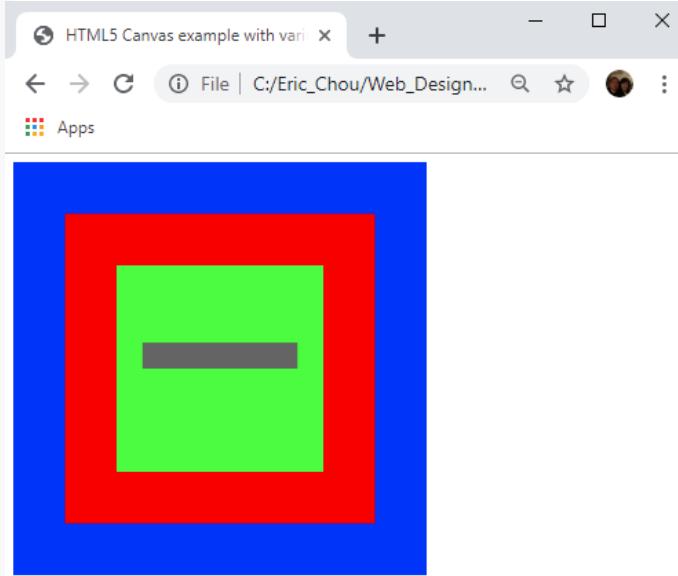


Demonstration Program

STARTER2.HTML



```
2 ▼ <html>
3 ▼   <head>
4     <title>HTML5 Canvas example with various methods </title>
5 ▼   <script>
6     window.onload=function() {
7       var canvas=document.getElementById("canvas");
8       var ctx=canvas.getContext('2d');
9       ctx.fillStyle='rgb(0,0,255)';
10      ctx.fillRect(0,0,400,400);
11      ctx.fillStyle='rgb(255,0,0)';
12      ctx.fillRect(50,50,300,300);
13      ctx.fillStyle='rgb(0,255,0)';
14      ctx.fillRect(100,100,200,200);
15      ctx.fillStyle='rgb(100,100,100)';
16      ctx.fillRect(125,175,150,25);
17    }
18  </script>
19 </head>
20 ▼ <body>
21   <div>
22     <canvas id="canvas" width="400" height="400"></canvas>
23   </div>
24 </body>
25 </html>
```



1. window.onload function.
allows the init() function to
be put in the head section.
2. Use id="canvas" to get the
canvas of current interest.
3. **ctx** (paintbrush) is like g in
java.



Setting Up Canvas

LECTURE 2



Setting Up Canvas

- To set up a canvas for drawing, you must add a <canvas> tag in HTML and assign a 2D drawing context to it. All the drawing operations are performed in the context

The <canvas> element

- In your HTML, include the following codes that define the canvas element, giving it a width and height.

```
<canvas id="myCanvas" height="300" width="400">  
</canvas>
```

- If a width or height is not specified, the default width of 300 pixels and the default height of 150 pixels are used. The canvas is initially empty and transparent.



The rendering context

- <canvas> creates a fixed-size drawing surface that exposes one or more rendering contexts, which are used to create and manipulate the content shown. We'll focus on the 2D rendering context. Other contexts may provide different types of rendering; for example, WebGL uses a 3D context ("experimental-webgl") based on OpenGL ES.
- The canvas is initially blank. To display something, a script first needs to access the rendering context and draw on it. The <canvas> element has a method called **getContext()**, used to obtain the rendering context and its drawing functions. **getContext()** takes one parameter, the type of context. For 2D graphics, such as those covered by this tutorial, you specify "2d".



Other Context

- The `getContext()` method returns a drawing context on the canvas, according to the type that you pass as parameter.
- Valid values are
 - `2d`, the one we'll use
 - `webgl` to use WebGL version 1
 - `webgl2` to use WebGL version 2
 - `bitmaprenderer` to use with [ImageBitmap](#)
- Based on the context type, you can pass a second parameter to `getContext()` to specify additional options.
- In the case of the 2d context, we basically have one parameter we can use in all browsers, and it's `alpha`, a boolean that defaults to true. If set to false, the browser knows the canvas does not have a transparent background and can speed up rendering.



Get DOM Canvas Object and Context Object

```
var canvas = document.getElementById('MyCanvas');  
var ctx = canvas.getContext('2d');
```

- The first line retrieves the DOM node for the <canvas> element by calling the document.getElementById() method. Once you have the element node, you can access the drawing context using its getContext() method which returns an object that provides methods and properties for drawing and manipulating images and graphics on a canvas element in a document.

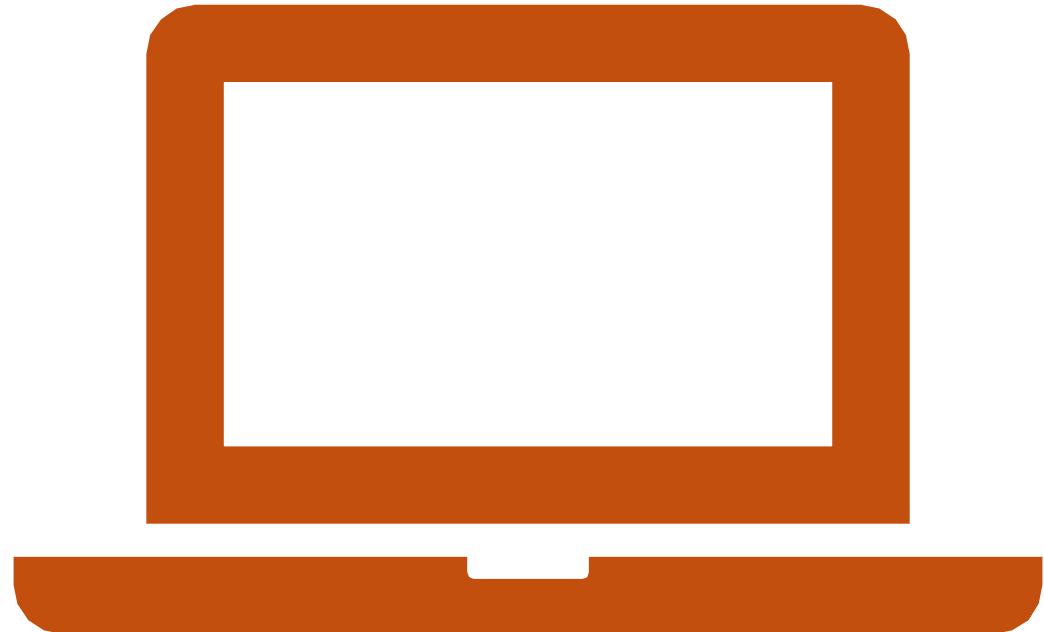




Checking for support

- The fallback content is displayed in browsers which do not support <canvas>. Through JavaScript you can check for support programmatically by simply testing for the presence of the getContext() method.
- If a coordinate system lies on your browser window, then most top-left position denotes (0,0). Moving right increases the value of x and moving down increases the value of y. So if no margin is set on the body of the HTML document, browser window's (0,0) coincides that of canvas's.





Demonstration Program

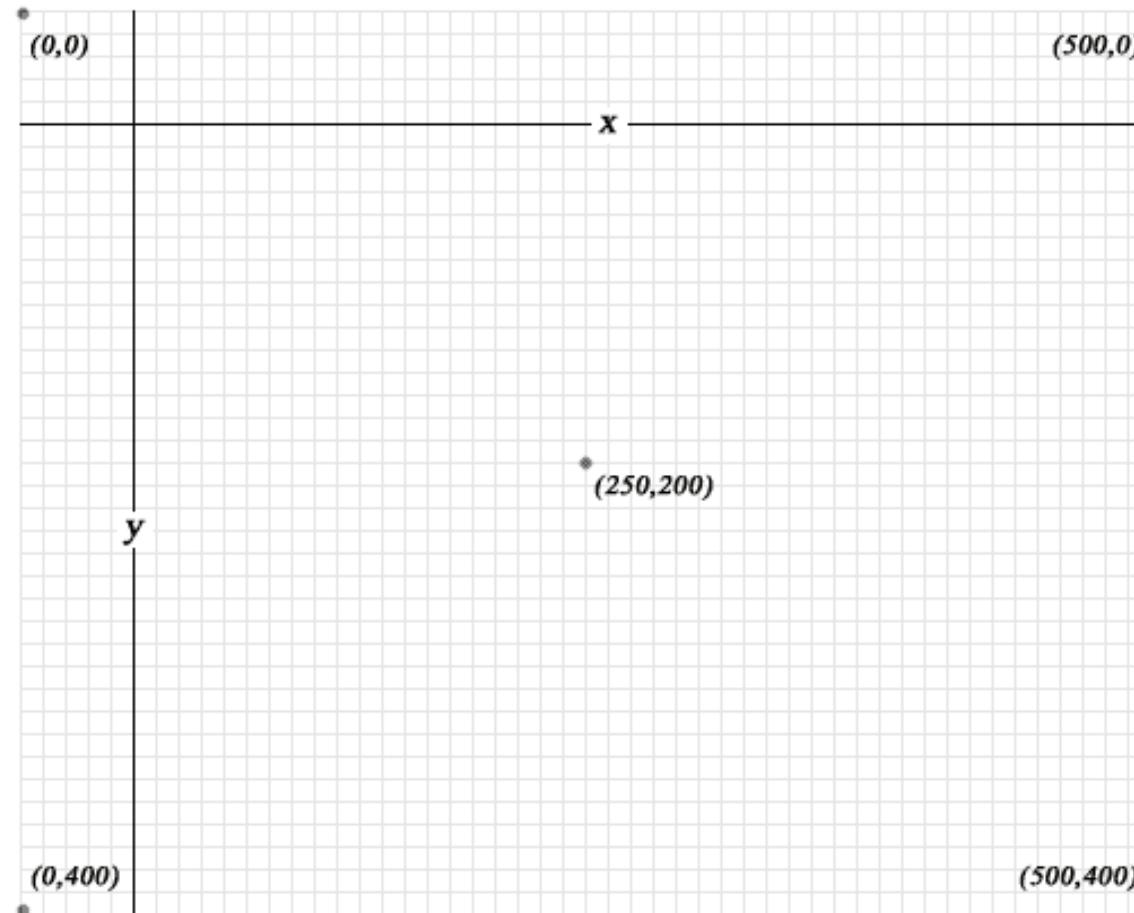
STARTER3.HTML

```
2 ▼ <html>
3 ▼   <head>
4     <meta charset=utf-8 />
5     <title>Draw a line</title>
6     <style>canvas{ background-color: beige}</style>
7   </head>
8 ▼   <body>
9     <canvas id="canvas" width="500" height="400"></canvas>
10 ▼    <script>
11       var canvas = document.getElementById('canvas');
12       //Always check for properties and methods, to make sure your
13       code doesn't break in other browsers.
14       if (canvas.getContext)
15       {
16         var ctx = canvas.getContext('2d');
17         // drawing code here
18         alert("My Browser support Canvase");
19       }
20       else
21       {
22         // canvas-unsupported code here
23         alert("My Browser doesn't support Canvase");
24       }
25     </script>
26   </body>
27 </html>
```





Pictorial Presentation of the above Canvas :





Full Page Canvas

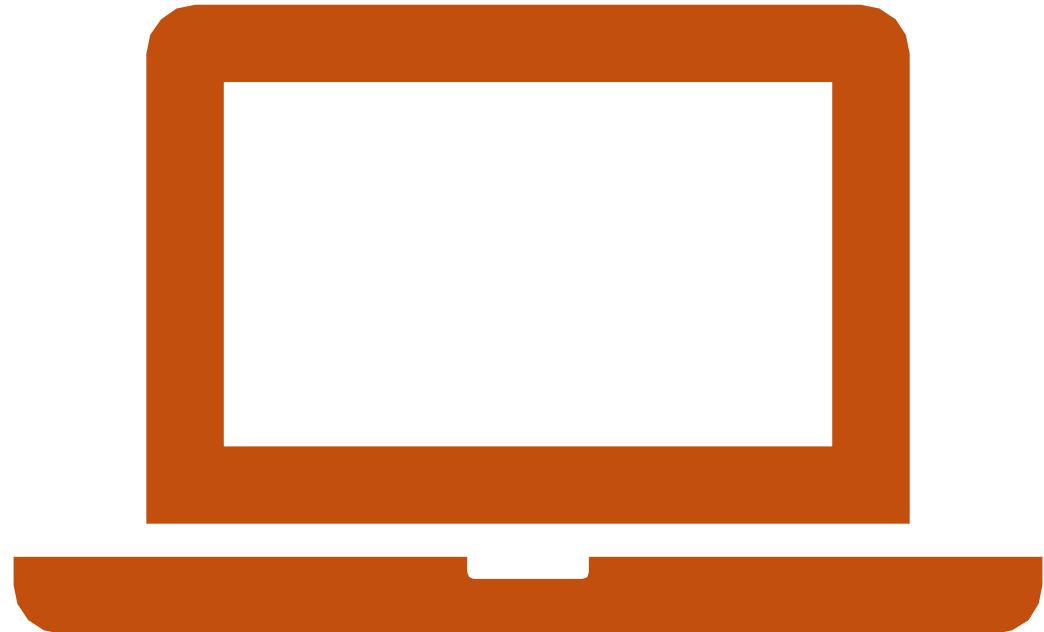
LECTURE 3



Set Up by querySelect();

- Our canvas is now reachable from JavaScript using the DOM Selectors API, so we can use **document.querySelector()**:

```
const canvas = document.querySelector('canvas')
```



Demonstration Program

STARTER4.HTML



Demo Program: starter4.html

```
2 ▼ <html>
3 ▼   <head>
4     <meta charset=utf-8 />
5     <title>Full Page</title>
6 ▼   <style>
7     canvas{
8       border: 1px solid black;
9       width:100%;
10      height: 100%;
11    }
12  </style>
13 </head>
```



```
14 ▼    <body>
15        <canvas id="canvas" width="500" height="400"></canvas>
16 ▼    <script>
17 ▼        function paint(g){
18            g.fillStyle='rgb(255,0,0)';
19            g.fillRect(50,50,300,300);
20            g.fillStyle='rgb(0,255,0)';
21            g.fillRect(100,100,200,200);
22            g.fillStyle='rgb(100,100,100)';
23        }
24        var canvas = document.querySelector('canvas');
25        var ctx;
26        canvas.width = window.innerWidth;
27        canvas.height = window.innerHeight;
28        if (canvas.getContext)
29            ctx = canvas.getContext('2d');
30        {   paint(ctx);
31        }
32    </script>
33    </body>
34 </html>
```



Full Page

x

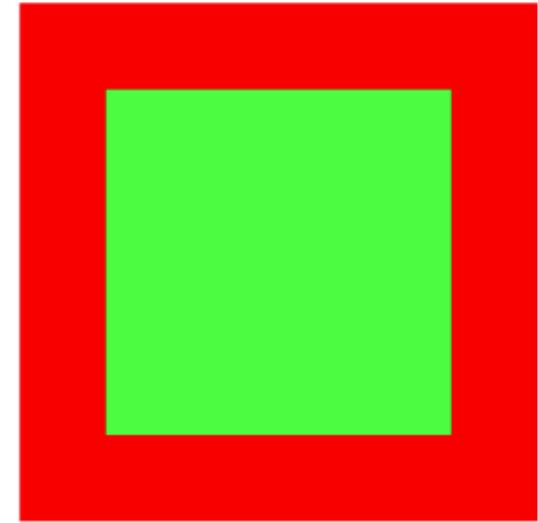
-

□

×



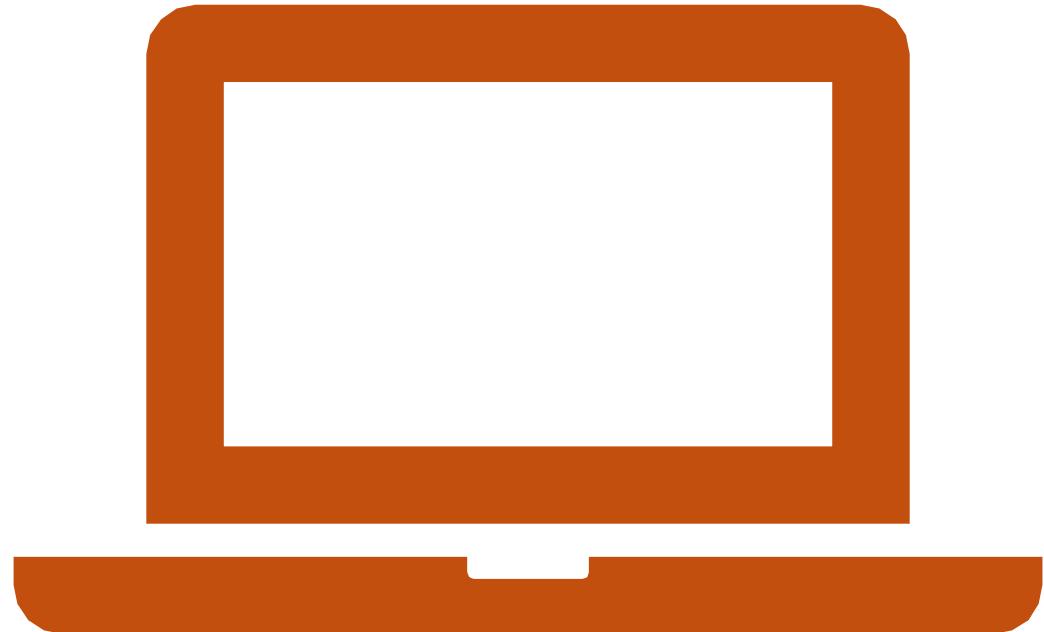
i File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Starter/A1%20Starter/starter4.html





Debouncing

LECTURE 4



Demonstration Program

STARTER5.HTML

```
const debounce = (func) => {
  let timer;
  return (event) => {
    if (timer) { clearTimeout(timer) }
    timer = setTimeout(func, 100, event);
  }
}

window.addEventListener('resize', debounce(() => {
  canvas.width = window.innerWidth;
  canvas.height = window.innerHeight;
  paint(ctx); // repaint
}))
```

Repaint the content

Debouncing

- If the window resizes we need to recalculate the canvas width as well, using a debounce to avoid calling too many times our canvas resizing (the resize event can be called hundreds of times as you move the window with the mouse, for example):



External JavaScript File

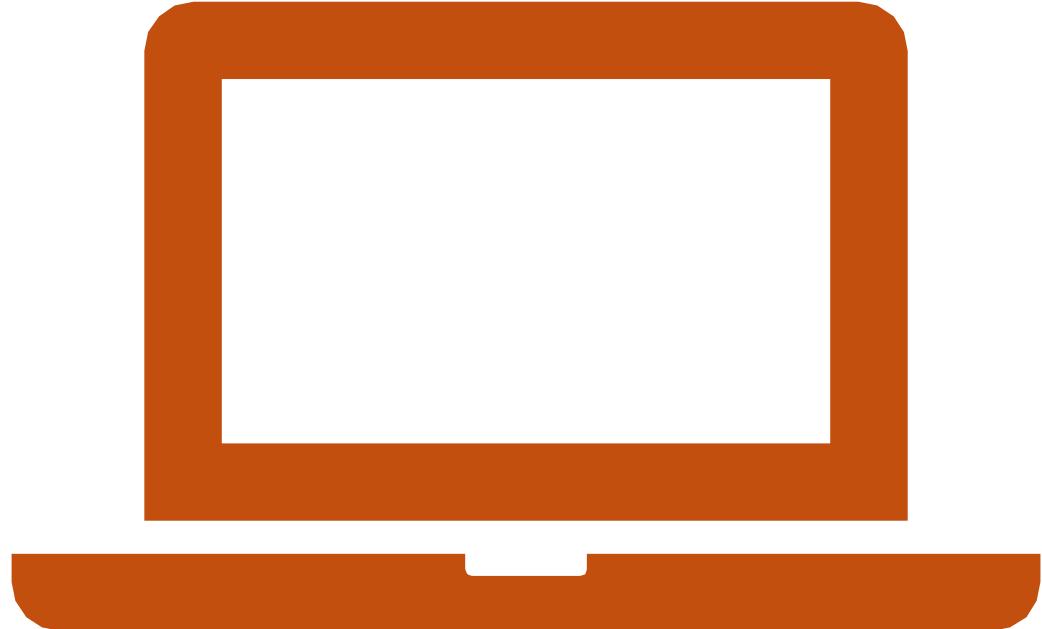
LECTURE 5



File Structure

- The canvas element can stay with a HTML page, while the JavaScript part can be put into a .js file.
- These two files are connected through the `getElementsById()` or `document.querySelector()`, and the context paint brush.

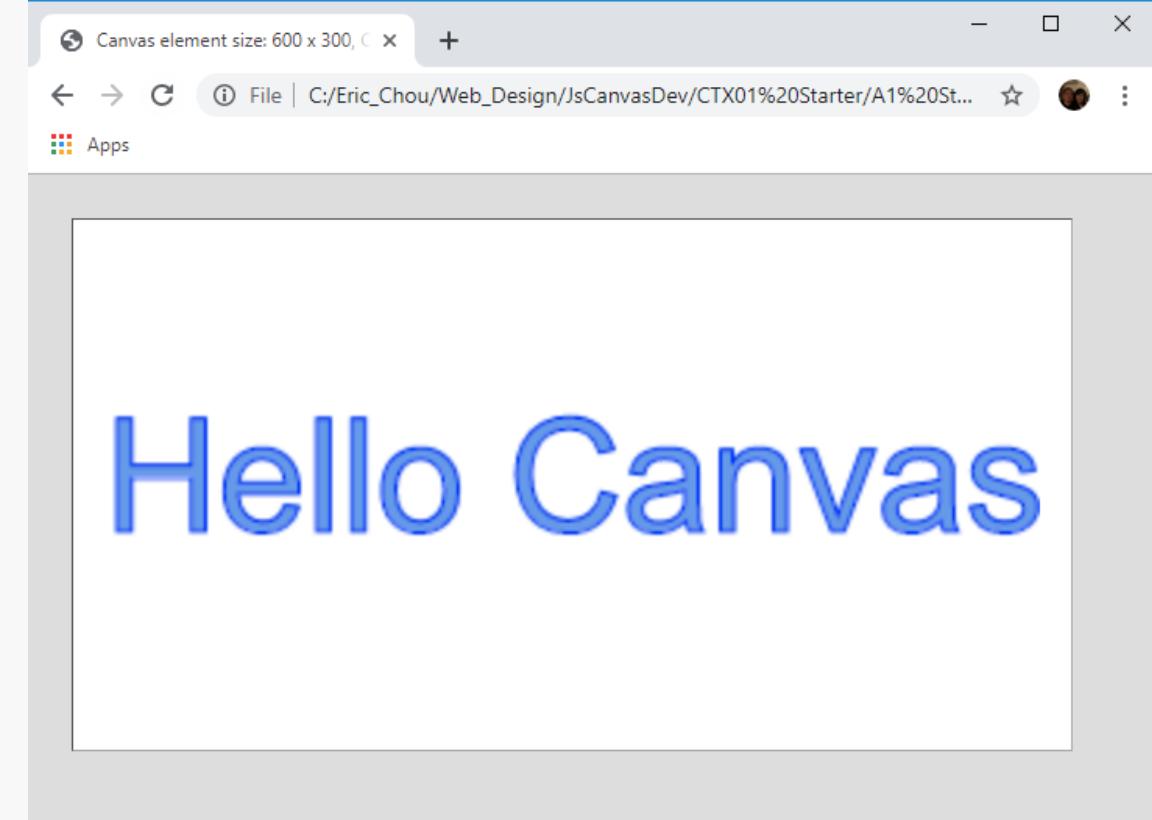




Demonstration Program

STARTER6.HTML + STARTER6.JS

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Canvas element size: 600 x 300,
5     Canvas drawing surface size: 300 x 150</title>
6 <style>
7     body {
8         background: #dddddd;
9     }
10    #canvas {
11        margin: 20px;
12        padding: 20px;
13        background: #ffffff;
14        border: thin inset #aaaaaa;
15        width: 600px;
16        height: 300px;
17    }
18 </style>
19 </head>
20 <body>
21    <canvas id='canvas'>
22        Canvas not supported
23    </canvas>
24    <script src='starter6.js'></script>
25 </body>
26 </html>
```





Demo Program: starter6.js

```
1 var canvas = document.getElementById('canvas'),  
2 context = canvas.getContext('2d');  
3 context.font = '38pt Arial';  
4 context.fillStyle = 'cornflowerblue';  
5 context.strokeStyle = 'blue';  
6 context.fillText('Hello Canvas', canvas.width/2 - 150, canvas.height/2 + 15);  
7 context.strokeText('Hello Canvas', canvas.width/2 - 150, canvas.height/2 + 15 );
```

Hello Canvas





Simple Text Message

- After obtaining a reference to the canvas's context, the JavaScript sets the context's font, **fillStyle**, and **strokeStyle** attributes and fills and strokes the text that you see in HTML page. The **fillText()** method fills the characters of the text using **fillStyle**, and **strokeText()** strokes the outline of the characters with **strokeStyle**. The **fillStyle** and **strokeStyle** attributes can be a CSS color, a gradient, or a pattern.
- We briefly discuss those attributes in Section 1.2.1, “The 2d Context,” on p. 9 and take a more in-depth look at both the attributes and methods in Chapter 2.



Simple Text Message

- The **fillText()** and **strokeText()** methods both take three arguments: the **text** and an **(x, y)** location within the canvas to display the text. The JavaScript shown in Example 1.2 approximately centers the text with constant values, which is not a good general solution for centering text in a canvas. In Chapter 3, we will look at a better way to center text.
- Book Chapter numbers for **Core HTML5 Canvas, 2nd Edition**



Basic Elements

LECTURE 6



Draw elements to a canvas

With the context we can now draw elements.

We have several methods to do so. We can draw:

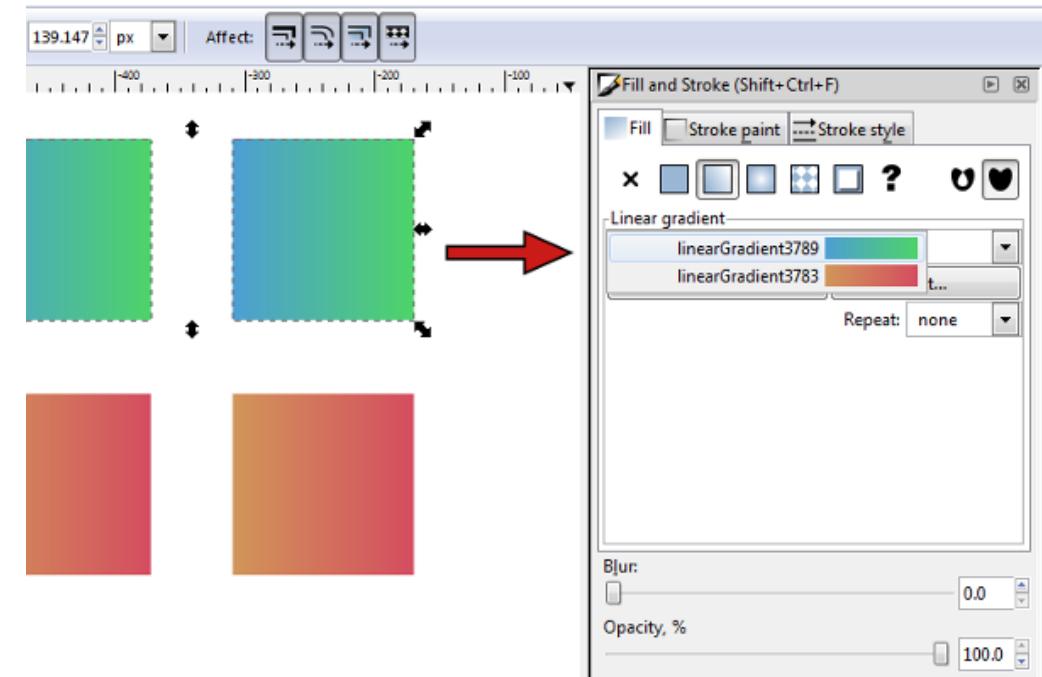
- text
- lines
- rectangles
- paths
- images

and for each of those elements we can alter the **fill**, the **stroke**, the **gradient**, the **pattern**, the **shadow**, **rotate** them, **scale** and perform a lot of operations.



Draw elements to a canvas

In this chapter, we only show basic elements. For more details, please check other chapters.





Rectangles

ACTIVITY





Rectangle

- Let's start with the simplest thing: a rectangle.
- The **fillRect(x, y, width, height)** method serves this purpose:

```
c.fillRect(100, 100, 100, 100)
```

SHAPES

RECTANGLE

To draw a rectangle:

rect(x, y, width, height)

fillRect(x, y, width, height)

strokeRect(x, y, width, height)

clearRect(x, y, width, height)





Demonstration Program

RECTANGLE.HTML + INIT.JS



Demo Program: rectangle.html+init.js

- init.js setup the canvas and the debouncing for the window.
- No re-paint function in this example.
- **canvas** and **ctx** is not a shared variable among two JavaScript sections.
- After the resizing the rectangle is of different shape.





Demo Program: rectangle.html+init.js

- This is going to draw a black rectangle of 100 x 100 pixels, starting from position x 100 and y 100.
- You can color the rectangle by using the `fillStyle()` method, passing any valid CSS color string:

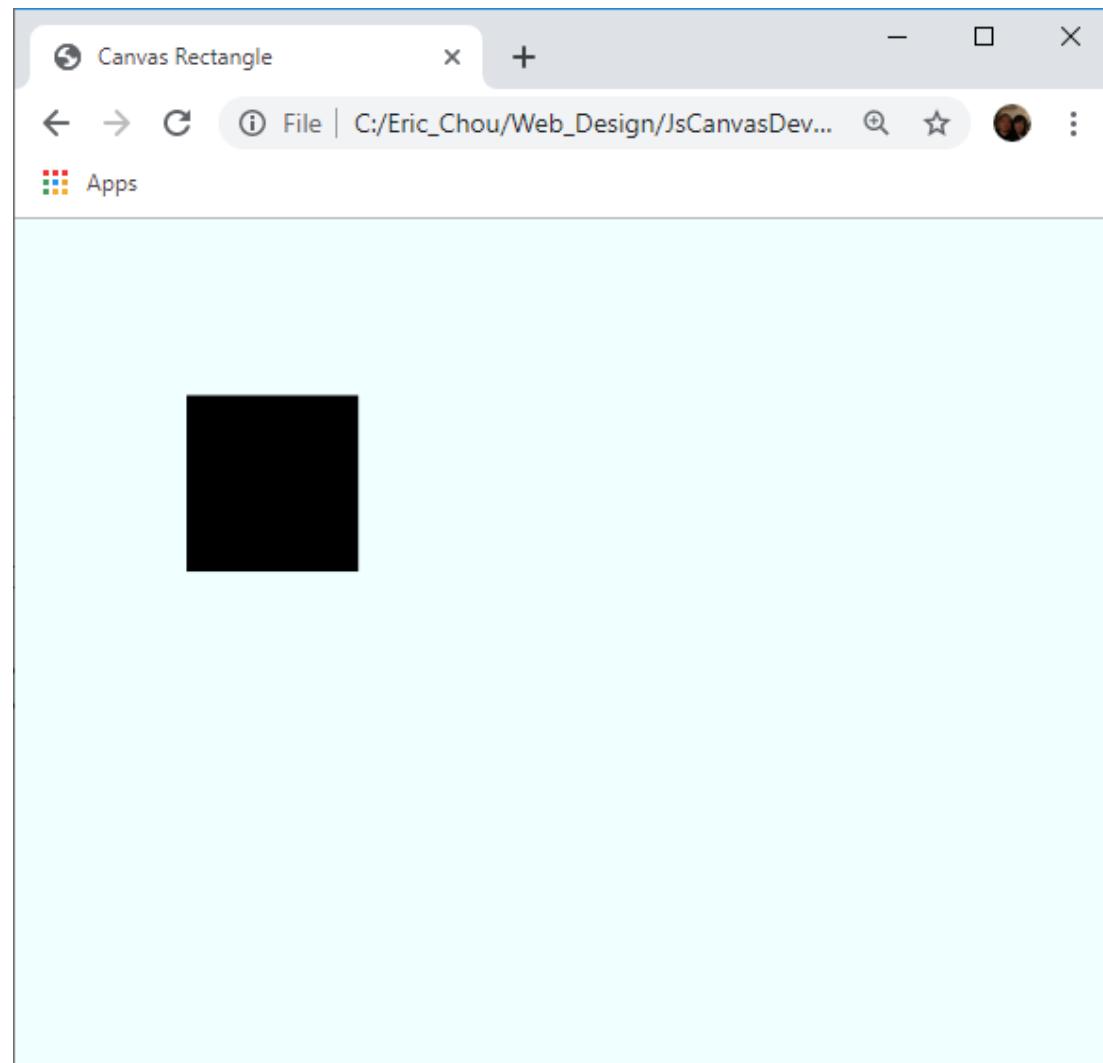
```
c.fillStyle = 'white'  
c.fillRect(100, 100, 100, 100)
```

```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:azure;
10    }
11   </style>
12   <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="640px" height="480px"></canvas>
16 ▼   <script>
17     var canvas = document.querySelector('canvas');
18     var ctx= canvas.getContext('2d');
19
20     ctx.fillStyle='black';
21     ctx.fillRect(100,100,100,100);
22   </script>
23 </body>
24 </html>
```



```
1 const canvas = document.querySelector('canvas');
2 const ctx;
3
4 canvas.width  = window.innerWidth;
5 canvas.height = window.innerHeight;
6 if (canvas.getContext)
7     {   ctx = canvas.getContext('2d');
8     }
9
10 ▼ const debounce = (func) => {
11     let timer;
12 ▼ return (event) => {
13     if (timer) { clearTimeout(timer) }
14     timer = setTimeout(func, 100, event);
15     }
16 }
17
18 ▼ window.addEventListener('resize', debounce(() => {
19     canvas.width = window.innerWidth;
20     canvas.height = window.innerHeight;
21 }))
```







Demo Program: rectangle2.html+init.js

- Same init.js file as rectangle.html
- Parameterized color and location for the rectangles

```
for (let i = 0; i < 60; i++) {
    for (let j = 0; j < 60; j++) {
        c.fillStyle = `rgb(${i * 5}, ${j * 5}, ${((i+j) * 50)})`
        c.fillRect(j * 20, i * 20, 10, 10)
    }
}
```

```
1▼ <html>
2▼   <head>
3     <title>Canvas Rectangle</title>
4▼   <style>
5     body{ margin:0; }
6▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11  </style>
12  <script type="javascript/text" src="init.js"></script>
13 </head>
14▼ <body>
15  <canvas id="canvas" width="640px" height="480px"></canvas>
16▼ <script>
17  var canvas = document.querySelector('canvas');
18  var ctx= canvas.getContext('2d');

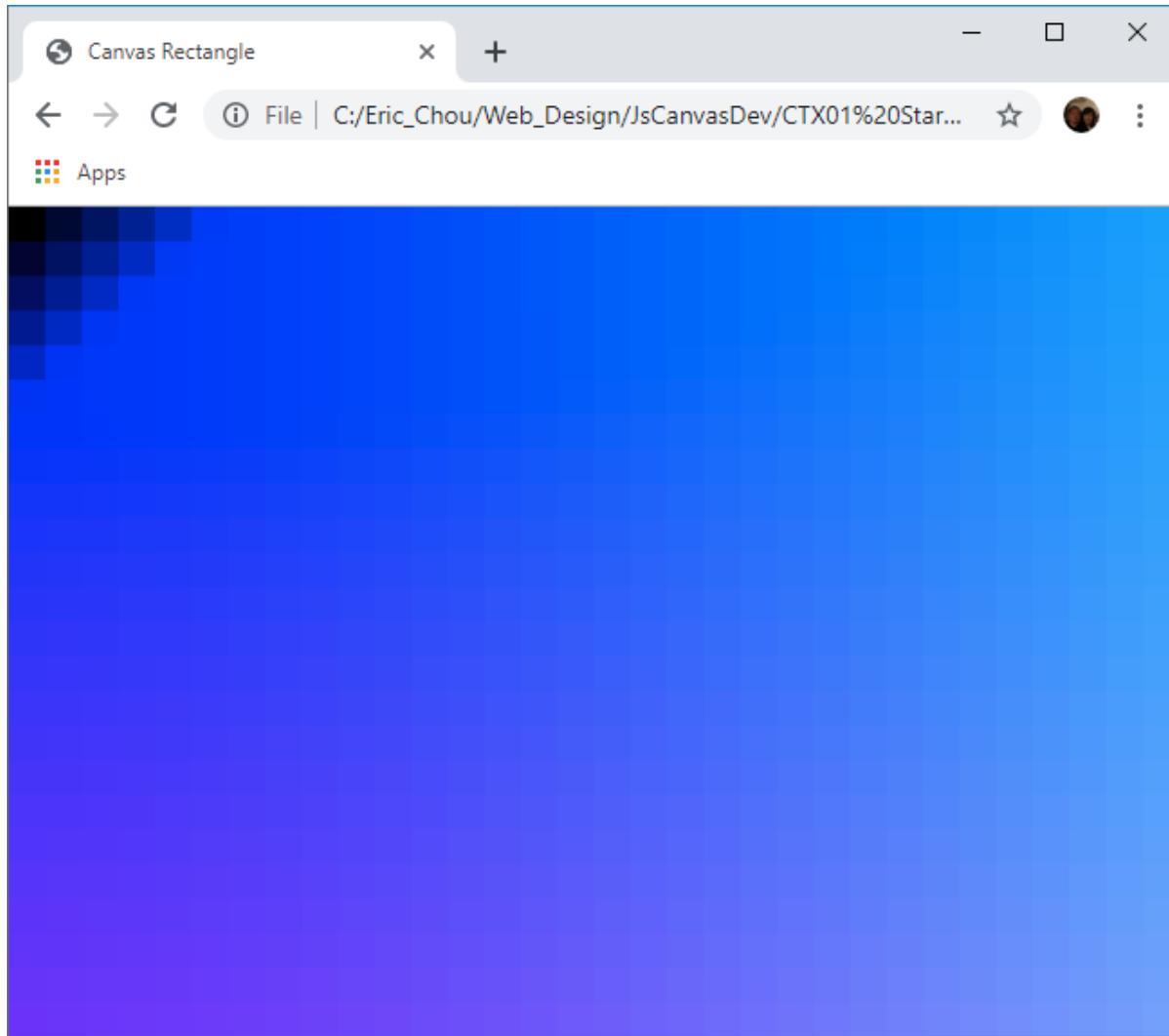
19
20▼  for (let i = 0; i < 60; i++) {
21▼    for (let j = 0; j < 60; j++) {
22      ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${((i+j) * 50)})`;
23      ctx.fillRect(j * 20, i * 20, 10, 10)
24    }
25  }
26 </script>
27 </body>
28 </html>
```





```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11  </style>
12  <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="640px" height="480px"></canvas>
16 ▼ <script>
17   var canvas = document.querySelector('canvas');
18   var ctx= canvas.getContext('2d');

19
20 ▼   for (let i = 0; i < 60; i++) {
21 ▼     for (let j = 0; j < 60; j++) {
22       ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${((i+j) * 50)})`;
23       ctx.fillRect(j * 20, i * 20, 20, 20)
24     }
25   }
26 </script>
27 </body>
28 </html>
```

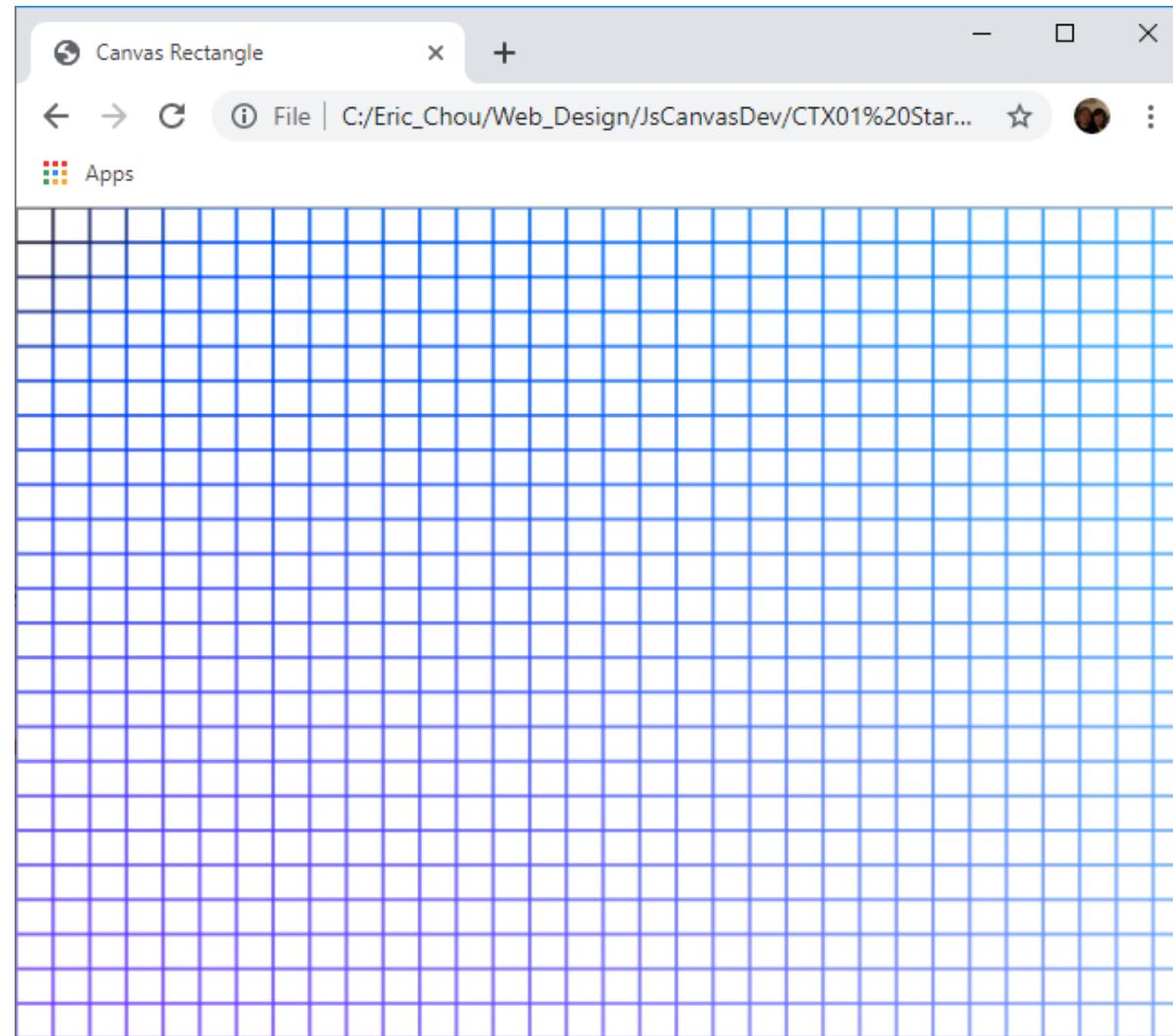


```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4     <style>
5       body{ margin:0; }
6       canvas{
7         width: 100%;
8         height: 100%;
9         background-color:white;
10      }
11    </style>
12    <script type="javascript/text" src="init.js"></script>
13  </head>
14 ▼ <body>
15   <canvas id="canvas" width="640px" height="480px"></canvas>
16 ▼ <script>
17   var canvas = document.querySelector('canvas');
18   var ctx= canvas.getContext('2d');

19
20 ▼   for (let i = 0; i < 60; i++) {
21 ▼     for (let j = 0; j < 60; j++) {
22       ctx.strokeStyle = `rgb(${i * 5}, ${j * 5}, ${i+j} * 50)`
23       ctx.strokeRect(j * 20, i * 20, 20, 20)
24     }
25   }
26 </script>
27 </body>
28 </html>
```

```
ctx.strokeStyle =
`rgb(${i * 5}, ${j * 5}, ${i+j} * 50)`

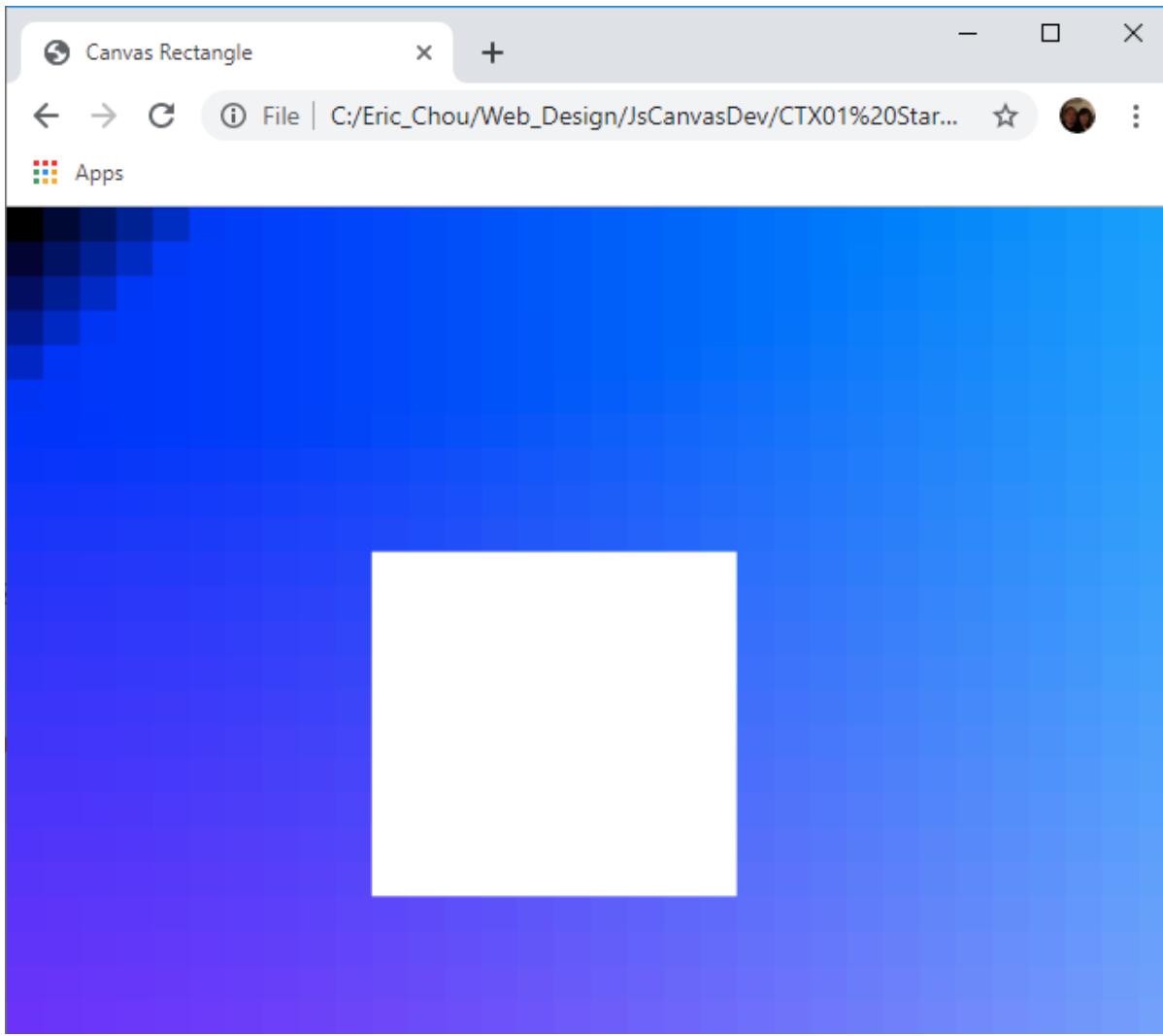
ctx.strokeRect(j * 20, i * 20, 20, 20)
```



clearRect() sets an area as transparent:

```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11   </style>
12   <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="640px" height="480px"></canvas>
16 ▼ <script>
17   var canvas = document.querySelector('canvas');
18   var ctx= canvas.getContext('2d');

19
20 ▼   for (let i = 0; i < 60; i++) {
21 ▼     for (let j = 0; j < 60; j++) {
22       ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${((i+j) * 50)})` 
23       ctx.fillRect(j * 20, i * 20, 20, 20)
24     }
25   }
26   ctx.clearRect(200, 200, 200, 200)
27 </script>
28 </body>
29 </html>
```





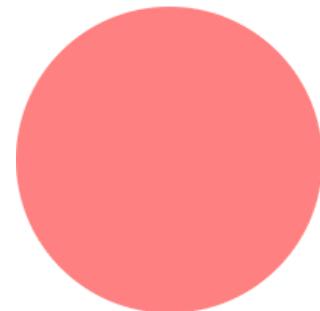
Drawing elements

Changing the colors

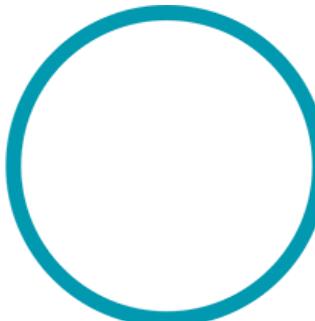
Use the `fillStyle` and `strokeStyle` properties to change the fill and stroke colors of any figure.

They accept any valid CSS color, including strings and RGB calculations:

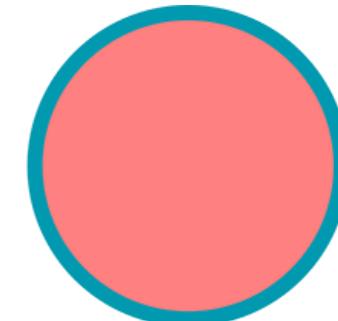
```
c.strokeStyle = `rgb(255, 255, 255)`
c.fillStyle = `white`
```



Fill



Stroke



Fill and stroke





Drawing elements

Rectangles

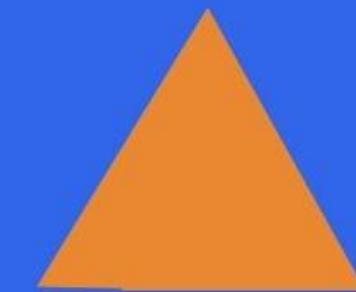
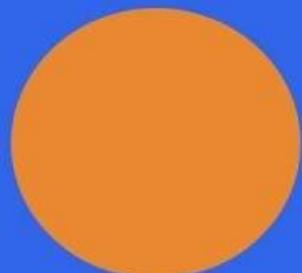
- `clearRect(x, y, width, height)`
- `fillRect(x, y, width, height)`
- `strokeRect(x, y, width, height)`

We saw `fillRect()` in the previous section. `strokeRect()` is similar in how it's called, but instead of filling a rect, it just draws the stroke using the current stroke style (which can be changed using the `strokeStyle` context property):

```
const c = canvas.getContext('2d')
for (let i = 0; i < 61; i++) {
  for (let j = 0; j < 61; j++) {
    c.strokeStyle = `rgb(${i * 5}, ${j * 5}, ${((i+j) * 50)})`
    c.strokeRect(j * 20, i * 20, 20, 20)
  }
}
```



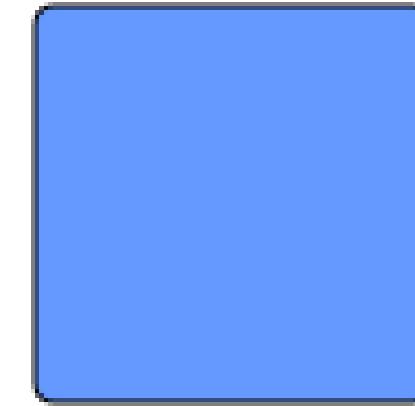
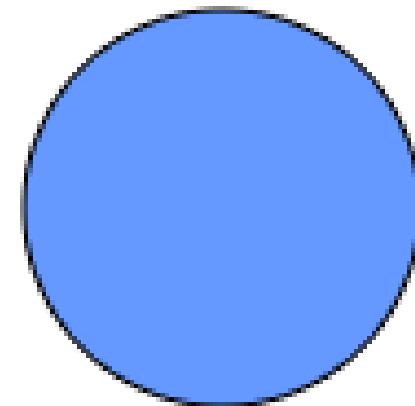
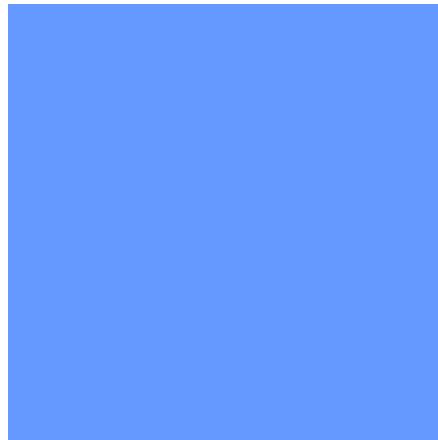
HTML-5 Canvas Shapes





12 Shapes to Extend HTML5 Canvas

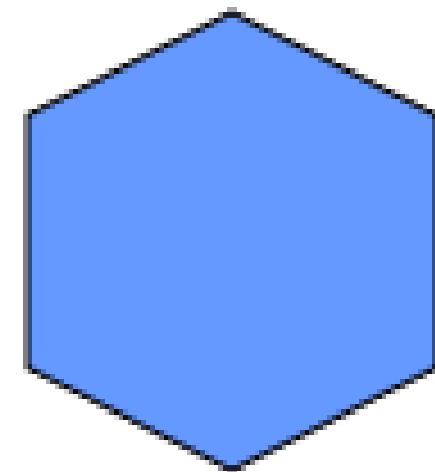
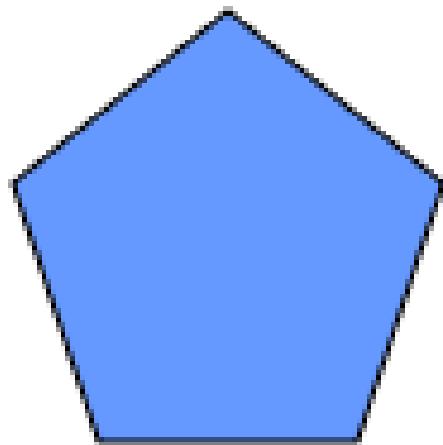
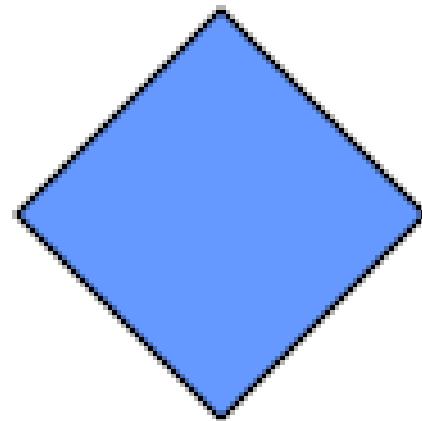
ctx.fillArea(x, y, color) **ctx.ellipse(x, y, w, h)** **ctx.roundedRect(x, y, w, h, radius)**





12 Shapes to Extend HTML5 Canvas

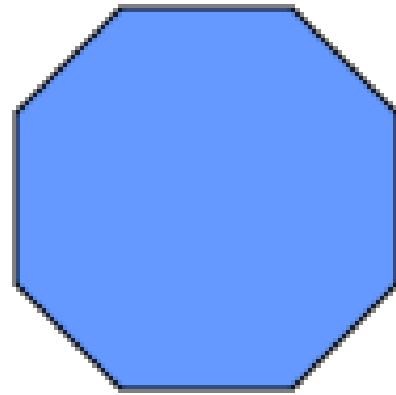
ctx.diamond(x, y, w, h) ctx.pentagon(x, y, w, h) ctx.hexagon(x, y, w, h)



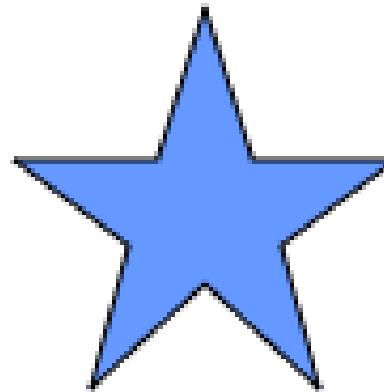


12 Shapes to Extend HTML5 Canvas

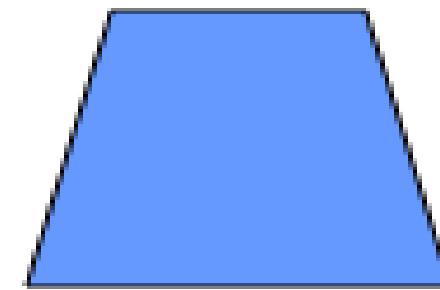
ctx.octagon(x, y, w, h)



ctx.star(x, y, w, h)



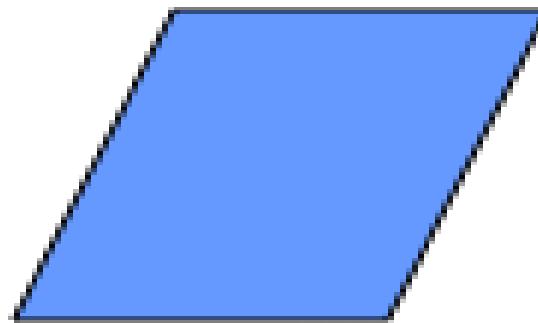
ctx.trapezoid(x, y, w, h)



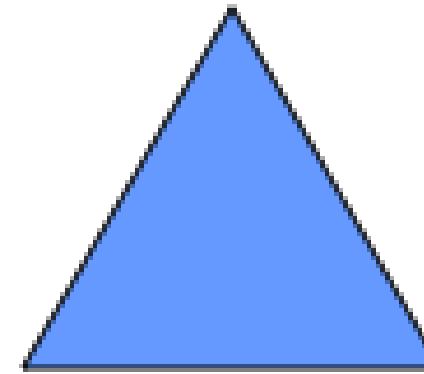


12 Shapes to Extend HTML5 Canvas

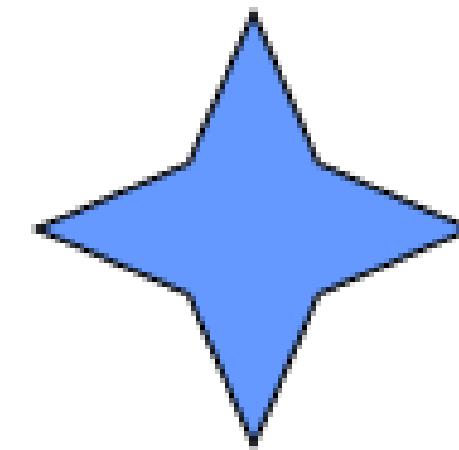
ctx.parallelgram(x, y, w, h)

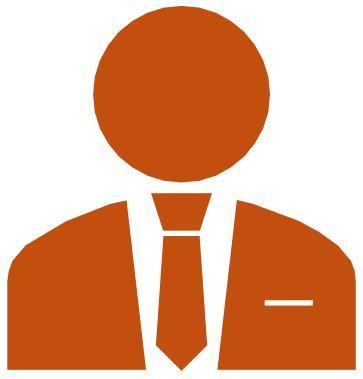


ctx.triangle(x, y, w, h)



ctx.ninjaStar(x, y, w, h)





Text

ACTIVITY



Text

TEXT

DRAWING TEXT:

```
fillText(text, x, y [, maxWidth])  
strokeText(text, x, y [, maxWidth])
```

STYLING TEXT:

```
font = value  
textAlign = value  
textBaseline = value  
direction = value
```

ADVANCED TEXT MEASUREMENT

```
measureText()
```

Drawing text is similar to rectangles. You have 2 methods

- `fillText(text, x, y)`
- `strokeText(text, x, y)`

which let you write text on the canvas.

x and y refer to the bottom-left corner. You change the font family and size using the font property of the canvas:

```
c.font = '148px Courier New'
```





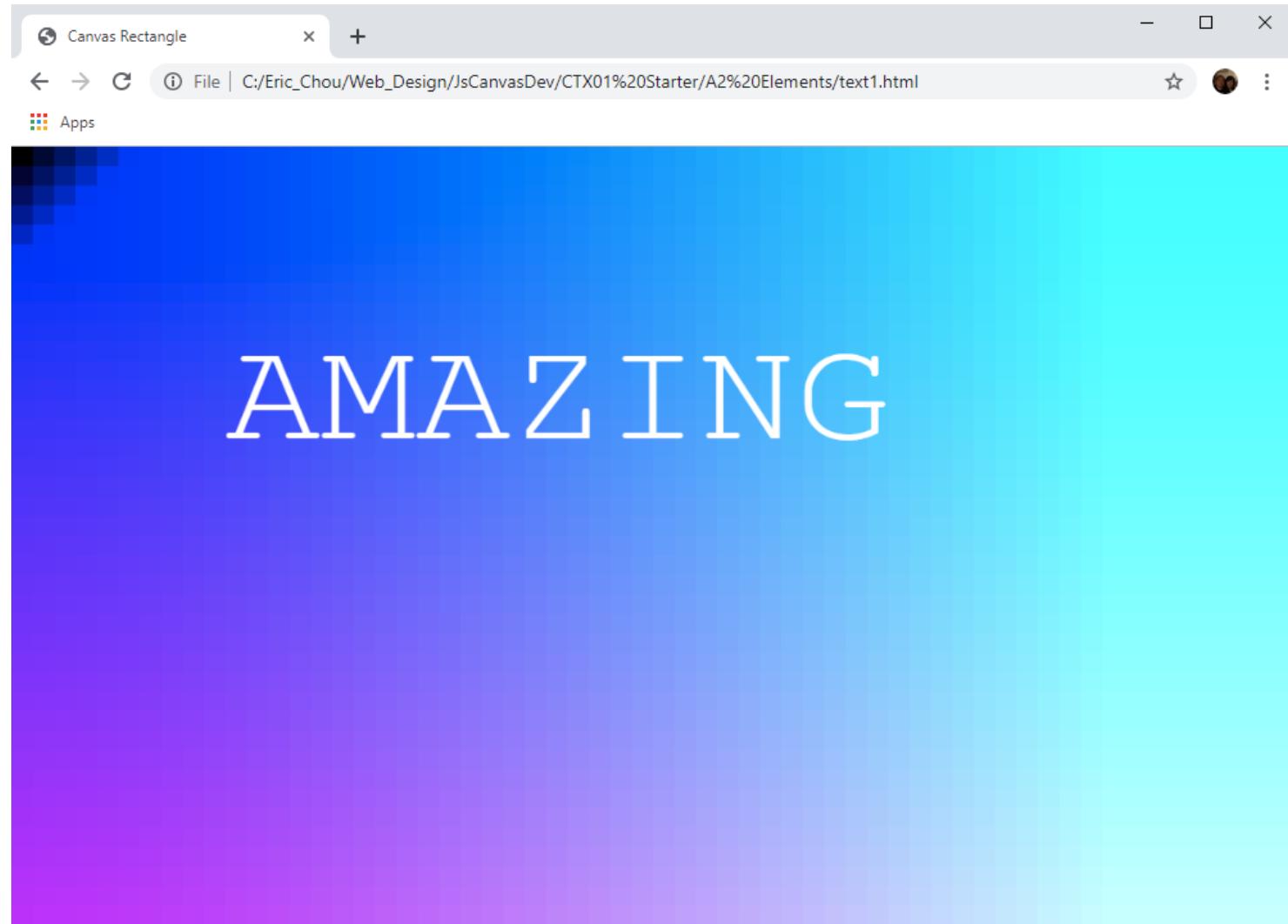
Text

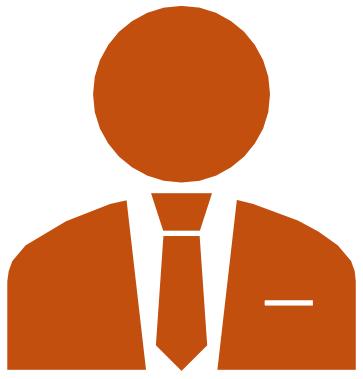
- There are other properties you can change, related to text (* = default):
 - textAlign (start*, end, left, right, center)
 - textBaseline (top, hanging, middle, alphabetic*, ideographic, bottom)
 - direction (ltr, rtl, inherit*)

```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11   </style>
12   <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="1200px" height="800px"></canvas>
16 ▼ <script>
17   var canvas = document.querySelector('canvas');
18   var ctx= canvas.getContext('2d');

19
20 ▼   for (let i = 0; i < 60; i++) {
21 ▼     for (let j = 0; j < 60; j++) {
22       ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`;
23       ctx.fillRect(j * 20, i * 20, 20, 20)
24     }
25   }
26   ctx.font = '148px Courier New';
27   ctx.fillText('AMAZING', 200, 300);
28 </script>
29 </body>
30 </html>
```







Lines

ACTIVITY



Lines

- To draw a line you first call the `beginPath()` method, then you provide a starting point with `moveTo(x, y)`, and then you call `lineTo(x, y)` to make the line to that new coordinates set. You finally call `stroke()`:

```
c.beginPath()  
c.moveTo(10, 10)  
c.lineTo(300, 300)  
c.stroke()
```

- The line is going to be colored according to the `c.strokeStyle` property value.



Polygon by Line

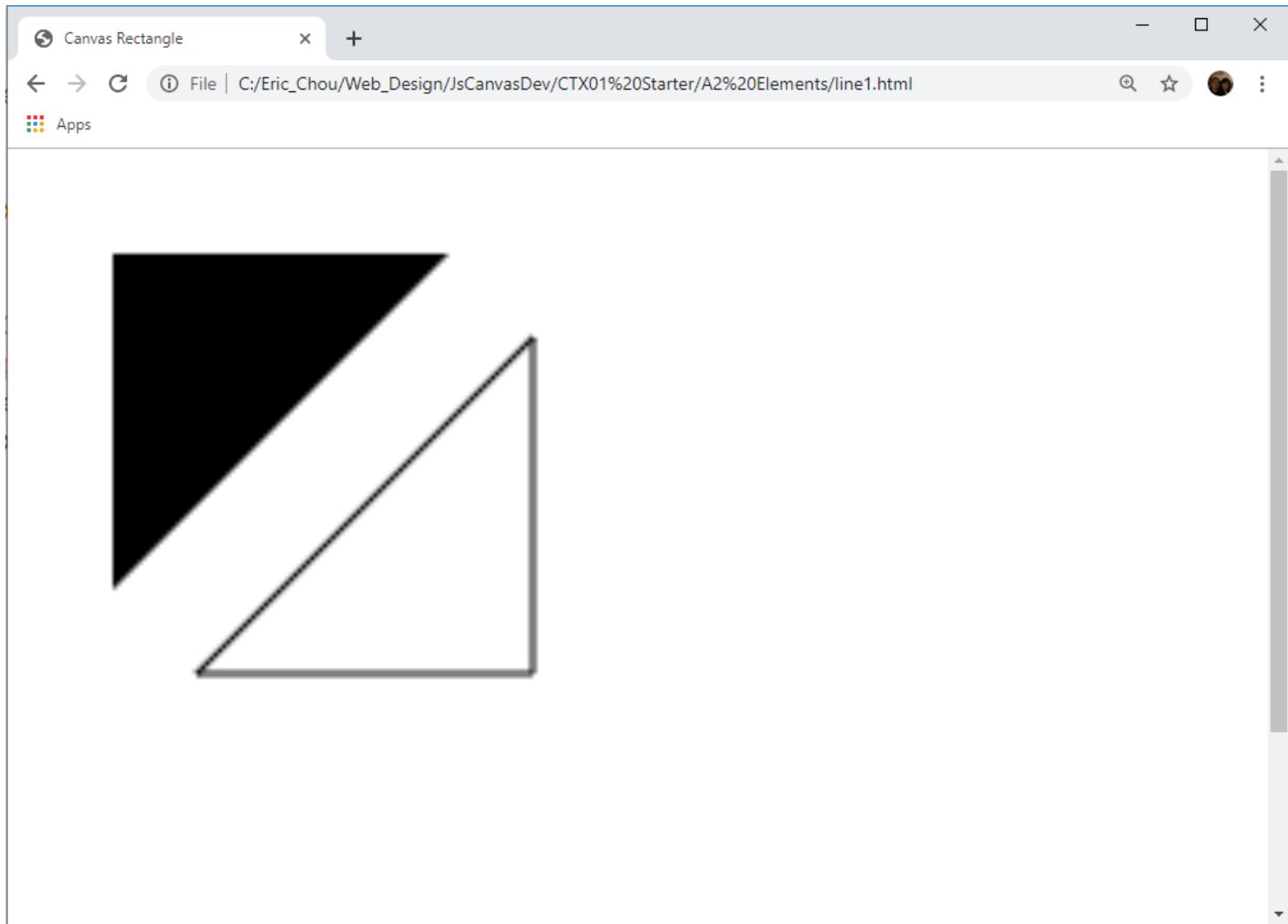
Filled triangle

```
ctx.beginPath();  
ctx.moveTo(25,25);           // First Point  
ctx.lineTo(105,25);  
ctx.lineTo(25,105);  
ctx.fill();                  // Fill or Stroke
```



```
<body>
  <canvas id="canvas" width="300px" height="240px"></canvas>
  <script type = "text/javascript">
    var canvas = document.querySelector('canvas');
    var ctx= canvas.getContext('2d');
    // Filled triangle
    ctx.beginPath();
    ctx.moveTo(25,25);
    ctx.lineTo(105,25);
    ctx.lineTo(25,105);
    ctx.fill();
    // Stroked triangle
    ctx.beginPath();
    ctx.moveTo(125,125);
    ctx.lineTo(125,45);
    ctx.lineTo(45,125);
    ctx.closePath();
    ctx.stroke();
  </script>
</body>
```







Path

ACTIVITY



Overview

Once you prepared the context you can actually instruct it to draw. You can draw complex shapes using "paths". A path is made out of draw primitives.

To draw a path you have to take the following steps:

1. `beginPath()`
2. `add` lines, curves, arcs, etc
3. (optionally) `closePath()`
4. `stroke()` or `fill()`





beginPath()

- Any new path must start with a beginPath() call on canvas context.
- beginPath() function instruct the canvas to start a new drawing session. This will actually erase previous path (if any) and creates a new one.

```
//begin a new path  
ctx.beginPath();
```

- Every time you call it, is like you place your pen above the canvas and you are about to write.
- Also initially the pen is not positioned anywhere....is "in the air"



moveTo()

- After you opened a new path you have to position your pen on canvas.
- As initially the pen is still "in the air", you must place it somewhere on the canvas.
- To do that you must use `moveTo()` method which will position the pen at the desired position.

```
//position the pen at (0, 0) coordinates  
ctx.moveTo(0, 0);
```

- Every time you call it is like you place your pen on the canvas and you are about to continue the drawing.
- If do not do that the first draw instruction will be ignored and used to position the pen.



Add lines, curves, arcs, etc

Once you have the path started you can:

- add lines
- add curves
- add arcs
- and other primitives





Add lines, curves, arcs, etc

Here is how to draw some primitives:

```
//draw a line to (100, 100)  
ctx.lineTo(100, 100);
```

```
//draw (quadratic) curve from last point (100, 100) to  
// (400, 100)  
ctx.bezierCurveTo(150, 50, 300, 150, 400, 100)
```

Every time you draw a primitive: line, curve, circle the pen will stop where the last drawn primitive stopped.

So, in our case, the curve will start from where the line ended.





closePath()

- After you added the primitives you wanted you must tell context to close the path. This means that the method will try to draw a line from last painted point in path to the initial point in path.

```
//close current path  
ctx.closePath();
```

- If the path was already closed or the path is a single point than this method is ignored.
- A special case appear when you call fill() - it will simply try to visually close the path automatically and fill it. So people think that fill() will close the path when it does not....it will do it only visually...see the real examples on fill();
- Any new primitive (line, curve, etc) added after will not be added to the path.



stroke()

- Stroke simply outline currently opened path, up to the last point.

```
//outline current path  
ctx.stroke();
```

- You can continue to add new primitives to current path and call `stroke()` again.
- Anyway, there is something bizarre about stroke and that is that `stroke()` will repaint the same path up the current point from the beginning, even if previous parts of the path were painted before. This will result in the outline of the path to become thicker and darker as if you use the pen to thicker a line.



fill()

- **fill()** simply fill current path with current fill color.

```
// fill path  
ctx.fill();
```





Style and Color

ACTIVITY



HTML canvas fillStyle Property

The **canvas fillStyle property** is used to **set or return the color, gradient, or pattern used to fill the drawing.**

Style:

```
context.fillStyle = color | gradient | pattern;
```





Property Value

- **color:** It is used to set the filled color of drawing. The default value of canvas fillStyle property is black.
- **gradient:** It is used to set the gradient object to fill the drawing. The gradient object are linear or radial.
- **pattern:** It is used to set the pattern to fill the drawing.





Color

- Hexadecimal Code
- Color names
- `rgb(red, green, blue)`



Red colors

IndianRed	CD 5C 5C	205 92 92
LightCoral	F0 80 80	240 128 128
Salmon	FA 80 72	250 128 114
DarkSalmon	E9 96 7A	233 150 122
LightSalmon	FF A0 7A	255 160 122
Crimson	DC 14 3C	220 20 60
Red	FF 00 00	255 0 0
FireBrick	B2 22 22	178 34 34
DarkRed	8B 00 00	139 0 0

Pink colors

Pink	FF C0 CB	255 192 203
LightPink	FF B6 C1	255 182 193
HotPink	FF 69 B4	255 105 180
DeepPink	FF 14 93	255 20 147
MediumVioletRed	C7 15 85	199 21 133
PaleVioletRed	DB 70 93	219 112 147

Orange colors

LightSalmon	FF A0 7A	255 160 122
Coral	FF 7F 50	255 127 80
Tomato	FF 63 47	255 99 71
OrangeRed	FF 45 00	255 69 0
DarkOrange	FF 8C 00	255 140 0
Orange	FF A5 00	255 165 0

Yellow colors

Gold	FF D7 00	255 215 0
Yellow	FF FF 00	255 255 0
LightYellow	FF FF E0	255 255 224
LemonChiffon	FF FA CD	255 250 205
LightGoldenrodYellow	FA FA D2	250 250 210
PapayaWhip	FF EF D5	255 239 213
Moccasin	FF E4 B5	255 228 181
PeachPuff	FF DA B9	255 218 185
PaleGoldenrod	EE E8 AA	238 232 170
Khaki	F0 E6 8C	240 230 140
DarkKhaki	BD B7 6B	189 183 107

Purple colors

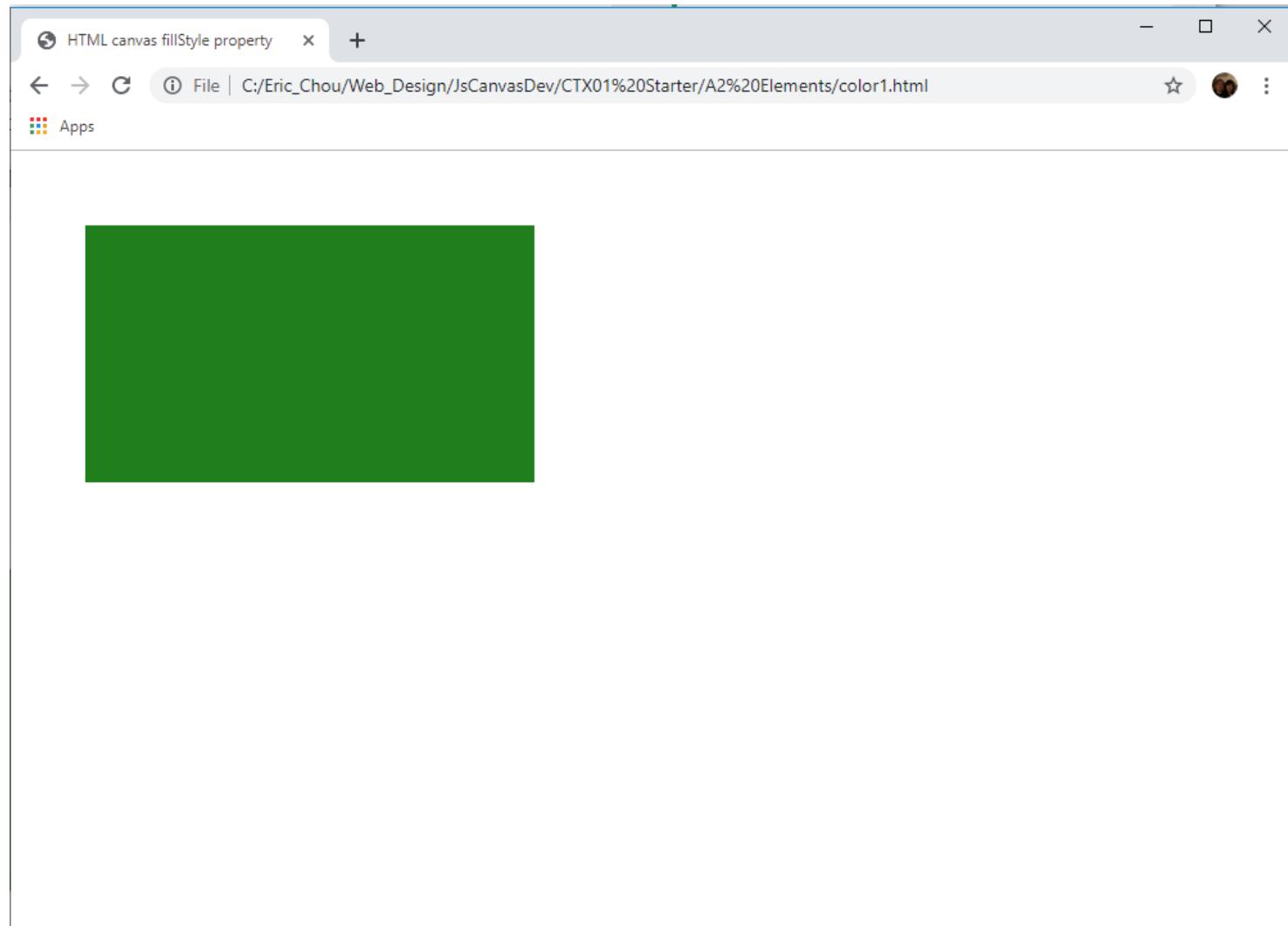
Lavender	E6 E6 FA	230 230 250
Thistle	D8 BF D8	216 191 216
Plum	DD A0 DD	221 160 221
Violet	EE 82 EE	238 130 238
Orchid	DA 70 D6	218 112 214
Fuchsia	FF 00 FF	255 0 255
Magenta	FF 00 FF	255 0 255
MediumOrchid	BA 55 D3	186 85 211
BlueViolet	8A 2B E2	138 43 226
DarkViolet	94 00 D3	148 0 211
DarkOrchid	99 32 CC	153 50 204
DarkMagenta	8B 00 8B	139 0 139
Purple	80 00 80	128 0 128
Indigo	4B 00 82	75 0 130
SlateBlue	6A 5A CD	106 90 205
DarkSlateBlue	48 3D 8B	72 61 139
MediumSlateBlue	7B 68 EE	123 104 238

Green colors		Blue/Cyan colors			Brown colors					White colors															
GreenYellow	AD FF 2F	173	255	47	Aqua	00	FF	FF	0	255	255	Cornsilk	FF	F8	DC	255	248	220	White	FF	FF	FF	255	255	255
Chartreuse	7F FF 00	127	255	0	Cyan	00	FF	FF	0	255	255	BlanchedAlmond	FF	EB	CD	255	235	205	Snow	FF	FA	FA	255	250	250
LawnGreen	7C FC 00	124	252	0	LightCyan	E0	FF	FF	224	255	255	Bisque	FF	E4	C4	255	228	196	Honeydew	F0	FF	F0	240	255	240
Lime	00 FF 00	0	255	0	PaleTurquoise	AF	EE	EE	175	238	238	NavajoWhite	FF	DE	AD	255	222	173	MintCream	F5	FF	FA	245	255	250
LimeGreen	32 CD 32	50	205	50	Aquamarine	7F	FF	D4	127	255	212	Wheat	F5	DE	B3	245	222	179	Azure	F0	FF	FF	240	255	255
PaleGreen	98 FB 98	152	251	152	Turquoise	40	E0	D0	64	224	208	BurlyWood	DE	B8	87	222	184	135	AliceBlue	F0	F8	FF	240	248	255
LightGreen	90 EE 90	144	238	144	MediumTurquoise	48	D1	CC	72	209	204	Tan	D2	B4	8C	210	180	140	GhostWhite	F8	F8	FF	248	248	255
MediumSpringGreen	00 FA 9A	0	250	154	DarkTurquoise	00	CE	D1	0	206	209	RosyBrown	BC	8F	8F	188	143	143	WhiteSmoke	F5	F5	F5	245	245	245
SpringGreen	00 FF 7F	0	255	127	CadetBlue	5F	9E	A0	95	158	160	SandyBrown	F4	A4	60	244	164	96	Seashell	FF	F5	EE	255	245	238
MediumSeaGreen	3C B3 71	60	179	113	SteelBlue	46	82	B4	70	130	180	Goldenrod	DA	A5	20	218	165	32	Belge	F5	F5	DC	245	245	220
SeaGreen	2E 8B 57	46	139	87	LightSteelBlue	B0	C4	DE	176	196	222	DarkGoldenrod	B8	86	0B	184	134	11	OldLace	FD	F5	E6	253	245	230
ForestGreen	22 8B 22	34	139	34	PowderBlue	B0	E0	E6	176	224	230	Peru	CD	85	3F	205	133	63	FloralWhite	FF	FA	F0	255	250	240
Green	00 80 00	0	128	0	LightBlue	AD	D8	E6	173	216	230	Chocolate	D2	69	1E	210	105	30	Ivory	FF	FF	F0	255	255	240
DarkGreen	00 64 00	0	100	0	SkyBlue	87	CE	EB	135	206	235	SaddleBrown	8B	45	13	139	69	19	AntiqueWhite	FA	EB	D7	250	235	215
YellowGreen	9A CD 32	154	205	50	LightSkyBlue	87	CE	FA	135	206	250	Sienna	A0	52	2D	160	82	45	Linen	FA	F0	E6	250	240	230
OliveDrab	6B 8E 23	107	142	35	DeepSkyBlue	00	BF	FF	0	191	255	Brown	A5	2A	2A	165	42	42	LavenderBlush	FF	F0	F5	255	240	245
Olive	80 80 00	128	128	0	DodgerBlue	1E	90	FF	30	144	255	Maroon	80	00	00	128	0	0	MistyRose	FF	E4	E1	255	228	225
DarkOliveGreen	55 6B 2F	85	107	47	CornflowerBlue	64	95	ED	100	149	237	Gray colors													
MediumAquamarine	66 CD AA	102	205	170	MediumSlateBlue	7B	68	EE	123	104	238	Gainsboro	DC	DC	DC	220	220	220	LightGrey	D3	D3	D3	211	211	211
DarkSeaGreen	8F BC 8F	143	188	143	RoyalBlue	41	69	E1	65	105	225	Silver	C0	C0	C0	192	192	192	DarkGray	A9	A9	A9	169	169	169
LightSeaGreen	20 B2 AA	32	178	170	MediumBlue	00	00	CD	0	0	205	Gray	80	80	80	128	128	128	DimGray	69	69	69	105	105	105
DarkCyan	00 8B 8B	0	139	139	DarkBlue	00	00	8B	0	0	139	LightSlateGray	77	88	99	119	136	153	SlateGray	70	80	90	112	128	144
Teal	00 80 80	0	128	128	Navy	00	00	80	0	0	128	Black	00	00	00	0	0	0							
					MidnightBlue	19	19	70	25	25	112														

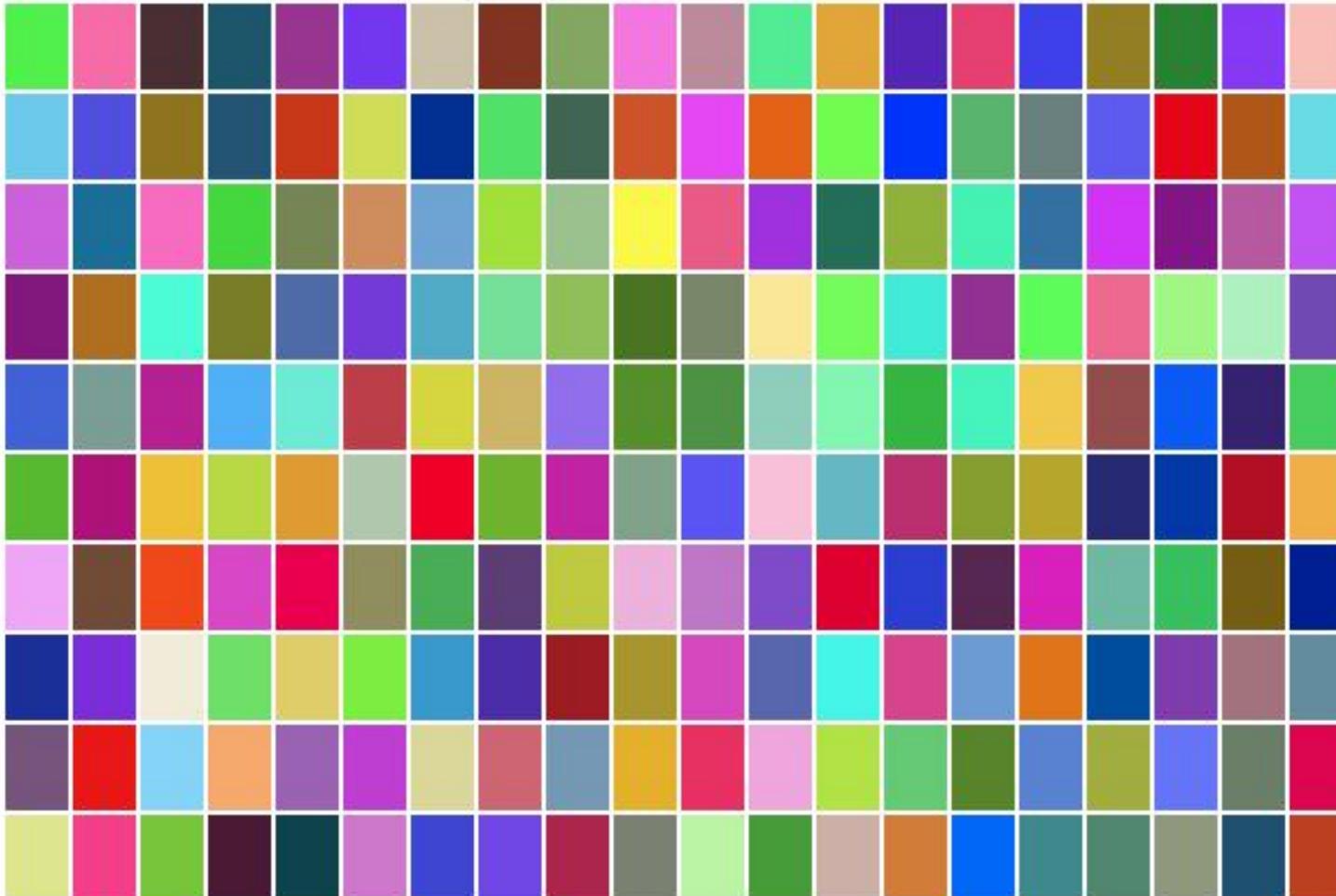


```
1  <!DOCTYPE html>
2  ▼ <html>
3  ▼     <head>
4  ▼         <title>
5             HTML canvas fillStyle property
6         </title>
7     </head>
8  ▼     <body>
9         <canvas id="GFG" width="500" height="300"></canvas>
10 ▼    <script>
11         var x = document.getElementById("GFG");
12         var contex = x.getContext("2d");
13         // set fillStyle color green.
14         contex.fillStyle = "green";
15         contex.fillRect(50, 50, 350, 200);
16         contex.stroke();
17     </script>
18     </body>
19 </html>
```





```
2 <head>
3     <title>HTML5 Canvas Colorful Tile Board</title>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <script type="javascript/text" src="init.js"></script>
7 </head>
```



```
8 <body>
9     <canvas width="600" height="400" id="canvas" ></canvas>
10    <script>
11        var canvas = document.getElementById("canvas");
12        var ctx = canvas.getContext("2d");
13        var CordinateX = 0;
14        var CordinateY = 0;
15        var Tilemaker = function(ctx,CordX,CordY,maxx,maxy){
16            var colorarr = ['1','2','3','4','5','6','7','8','9','0','a','b','c','d','e','f'];
17            var w = Math.floor(canvas.width/maxx)-2;
18            var h = Math.floor(canvas.height/maxy)-2;
19            var CordX0 = CordX;
20
21            for (var i = 0; i < maxy; i++) {
22                for (var j = 0; j < maxx; j++){
23                    var Color ='#';
24                    for(var k =0; k<6; k++){
25                        var r = Math.floor(Math.random()*16);
26                        Color = Color+colorarr[r];
27                    }
28                    ctx.fillStyle = Color;
29                    ctx.fillRect(CordX,CordY,w,h);
30                    CordX = CordX+w+2;
31                }
32                CordX = CordX0;
33                CordY = CordY+h+2;
34            }
35        }
36        Tilemaker(ctx,CordinateX,CordinateY,20,10);
37    </script>
38 </body>
```





Gradient

ctx.createLinearGradient(x0, y0, x1, y1);

Parameters

x0

The x-axis coordinate of the start point.

y0

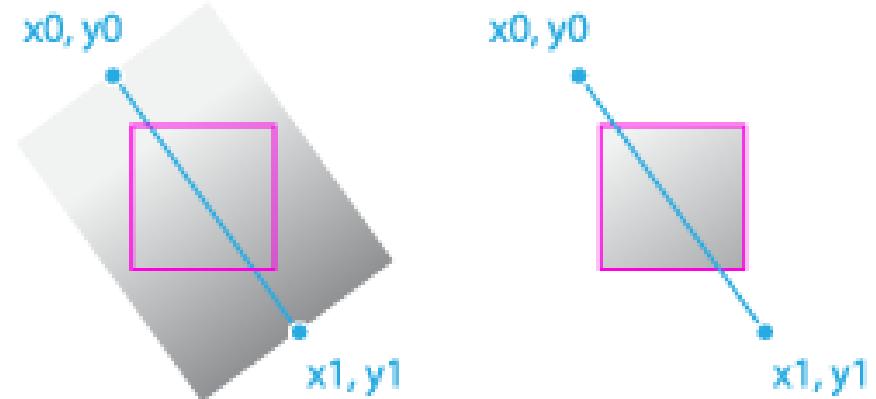
The y-axis coordinate of the start point.

x1

The x-axis coordinate of the end point.

y1

The y-axis coordinate of the end point.





Gradient

ctx.createLinearGradient(x0, y0, x1, y1);

Parameters

x0

The x-axis coordinate of the start point.

y0

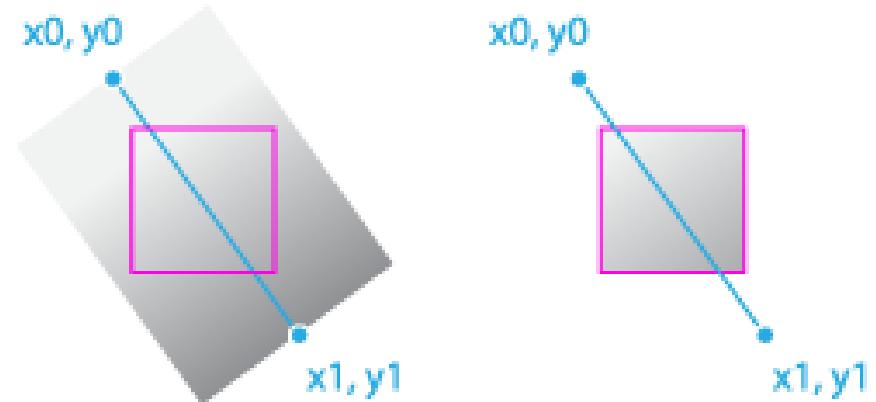
The y-axis coordinate of the start point.

x1

The x-axis coordinate of the end point.

y1

The y-axis coordinate of the end point.



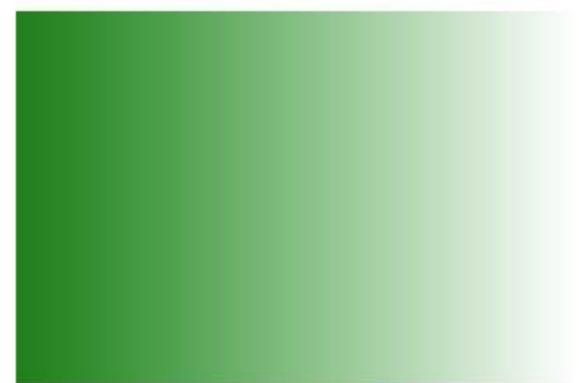


Gradient

```
var gr = context.createLinearGradient(50, 0, 350, 0);
  gr.addColorStop(0, "green");
  gr.addColorStop(1, "white");
context.fillStyle = gr;
context.fillRect(50, 50, 350, 200);
```



```
1  <!DOCTYPE html>
2  ▼ <html>
3  ▼ <head>
4  ▾   <title>
5      HTML canvas fillStyle property
6  </title>
7  </head>
8  ▼ <body>
9      <canvas id="GFG"
10         width="500"
11         height="300">
12  </canvas>
13 ▼   <script>
14     var x =
15       document.getElementById("GFG");
16     var contex =
17       x.getContext("2d");
18     var gr =
19       contex.createLinearGradient(50, 0, 350, 0);
20     gr.addColorStop(0, "green");
21     gr.addColorStop(1, "white");
22     contex.fillStyle = gr;
23     contex.fillRect(50, 50, 350, 200);
24     contex.stroke();
25   </script>
26 </body>
27 </html>
```



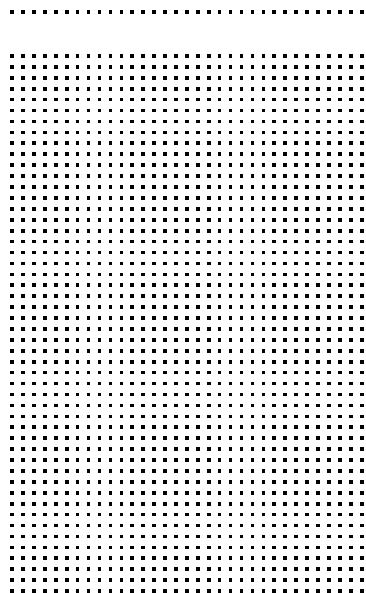
```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11   </style>
12   <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="640px" height="480px"></canvas>
16 ▼ <script>
17   var canvas = document.getElementById('canvas');
18   var ctx = canvas.getContext('2d');
19   // Create a linear gradient
20   // The start gradient point is at x=20, y=0
21   // The end gradient point is at x=220, y=0
22   var gradient = ctx.createLinearGradient(20,0, 220,0);
23   gradient.addColorStop(0, 'green');
24   gradient.addColorStop(.5, 'cyan');
25   gradient.addColorStop(1, 'green');
26   ctx.fillStyle = gradient;
27   ctx.fillRect(20, 20, 200, 100);
28 </script>
29 </body>
30 </html>
```



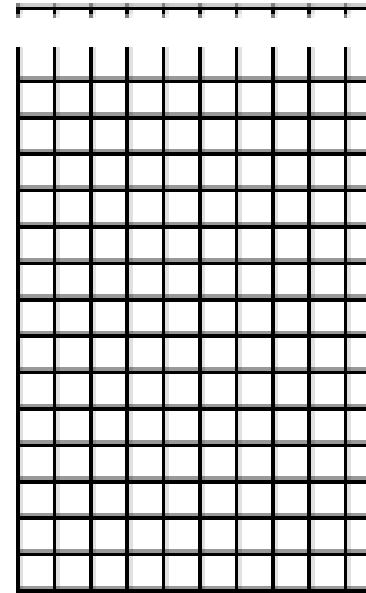


Pattern

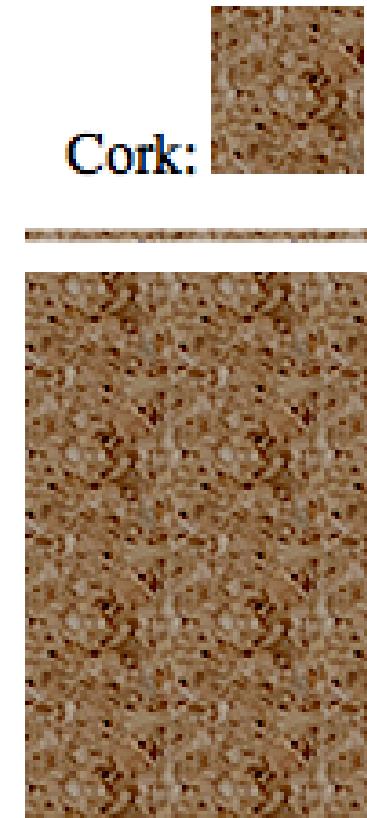
Dot: .



Block: L



Cork:





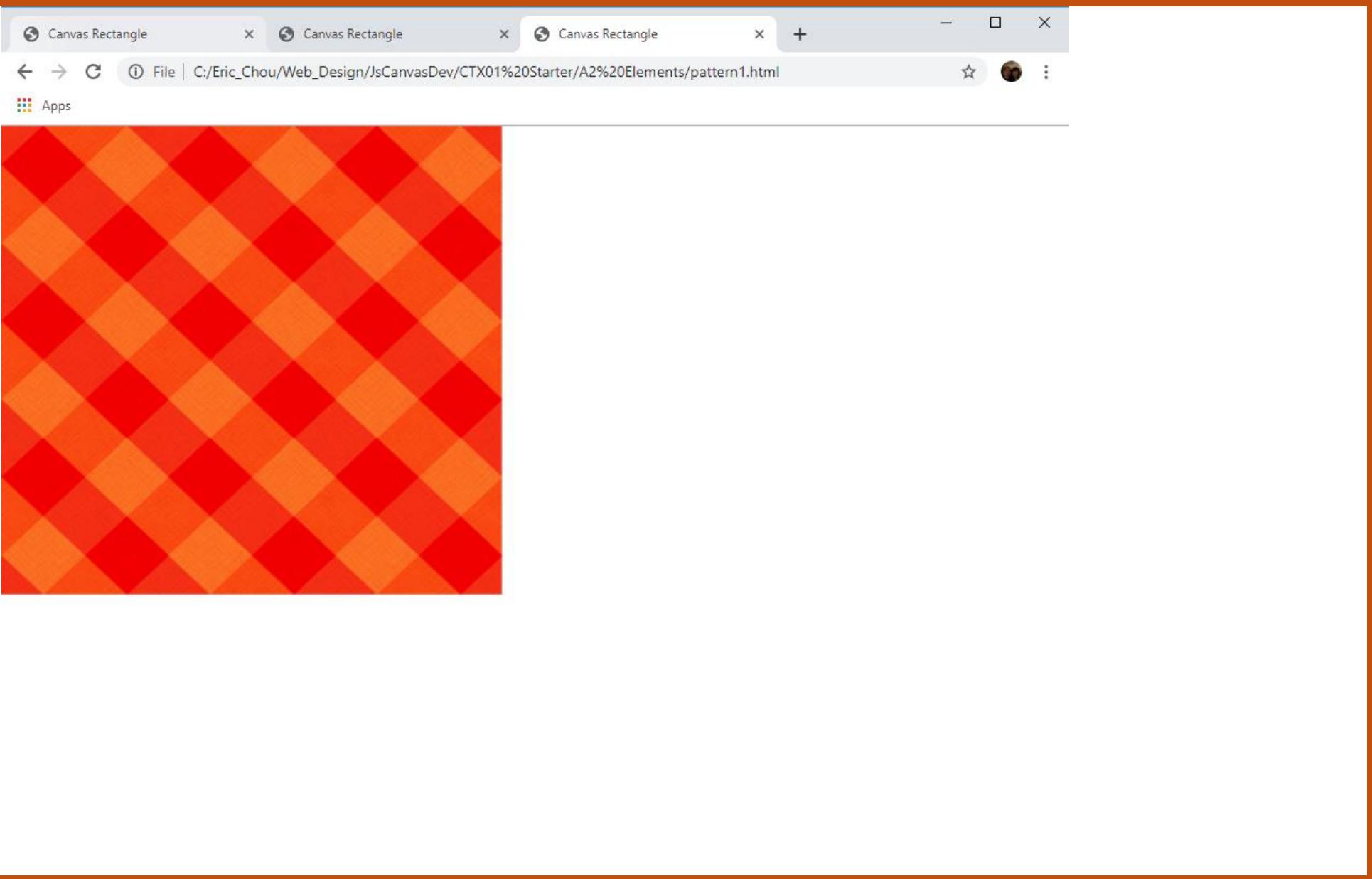
Pattern

```
var img = new Image();
img.src = "pattern.png";
img.onload = function drawPattern() {
    var p = ctx.createPattern(img, "repeat");
    ctx.fillStyle = p;
    ctx.fillRect(0, 0, 300, 300);
}
```



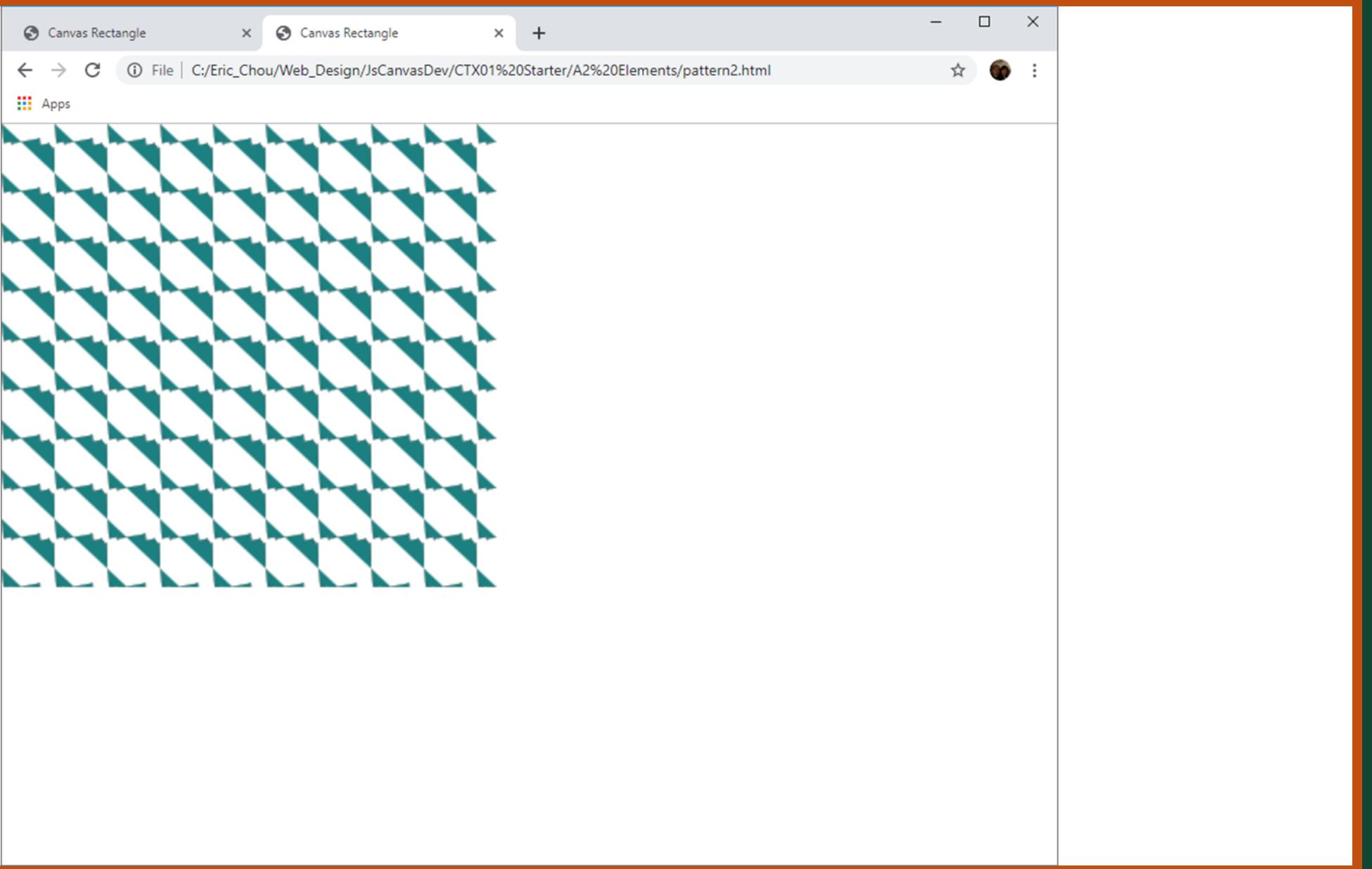
```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11  </style>
12  <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="640px" height="480px"></canvas>
16 ▼   <script>
17     var canvas = document.getElementById('canvas');
18     var ctx = canvas.getContext('2d');
19     var img = new Image();
20     img.src = "pattern.png";
21 ▼     img.onload = function drawPattern() {
22       var p = ctx.createPattern(img, "repeat");
23       ctx.fillStyle = p;
24       ctx.fillRect(0, 0, 300, 300);
25     }
26   </script>
27 </body>
28 </html>
```





```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11   </style>
12   <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="640px" height="480px"></canvas>
16 ▼ <script>
17   var canvas = document.getElementById('canvas');
18   var ctx = canvas.getContext('2d');
19   var img = new Image();
20   img.src = "bb.png";
21 ▼   img.onload = function drawPattern() {
22     var p = ctx.createPattern(img, "repeat");
23     ctx.fillStyle = p;
24     ctx.fillRect(0, 0, 300, 300);
25   }
26   </script>
27 </body>
28 </html>
```





STYLES & COLORS

COLORS:

fillStyle = color

strokeStyle = color

Transparency:

globalAlpha = transparencyValue

Line styles:

lineWidth = value

lineCap = type



lineJoin = type



round



bevel



miter

miterLimit = value

getLineDash()

setLineDash(segments)

lineDashOffset = value

GRADIENTS:

createLinearGradient(x1, y1, x2, y2)

createRadialGradient(x1, y1, r1, x2, y2, r2)

gradient.addColorStop(position, color)

PATTERNS:

createPattern(image, type)

SHADOWS:

shadowOffsetX = float

shadowOffsetY = float

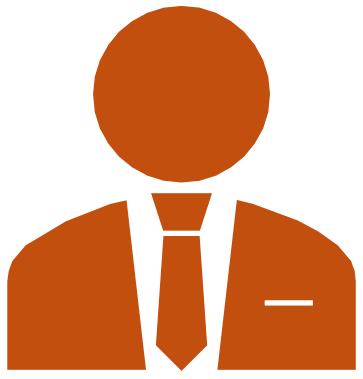
shadowBlur = float

shadowColor = color

CANVAS FILL RULES:

Nonzero-rule

Even-odd rule



Images

ACTIVITY





Image

- Image on Canvas is of completely different nature than the image on HTML node.
- The img tag element define an image node on a HTML page.
- The image on canvas is a colorful dot-matrix representation of an image. So, it can be overlapped, cleared or reloaded. It is not an element on HTML page.



Two ways of loading images

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
var img = new Image();
img.src = "rains.jpg";
img.onload = function drawPattern() {
    ctx.drawImage(img, 10, 10);
}


<script>
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');
    var img = document.getElementById("rains");
    window.onload = function(){
        ctx.drawImage(img, 10, 10);
    }

```



IMAGES

CANVAS API CAN USE ANY OF THE FOLLOWING DATA TYPES

HTMLImageElement
HTMLVideoElement
HTMLCanvasElement

GET AN IMAGE

from the same page:

from other domain:

use another canvas element:

Create images from scratch

Embedding an image via data: url.

Using frames from a video <video></video>

DRAW AN IMAGE:

drawImage(image, x, y)

SCALE AN IMAGE:

drawImage(image, x, y, width, height)

SLICE AN IMAGE:

drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)

CONTROL IMAGE SCALING BEHAVIOR

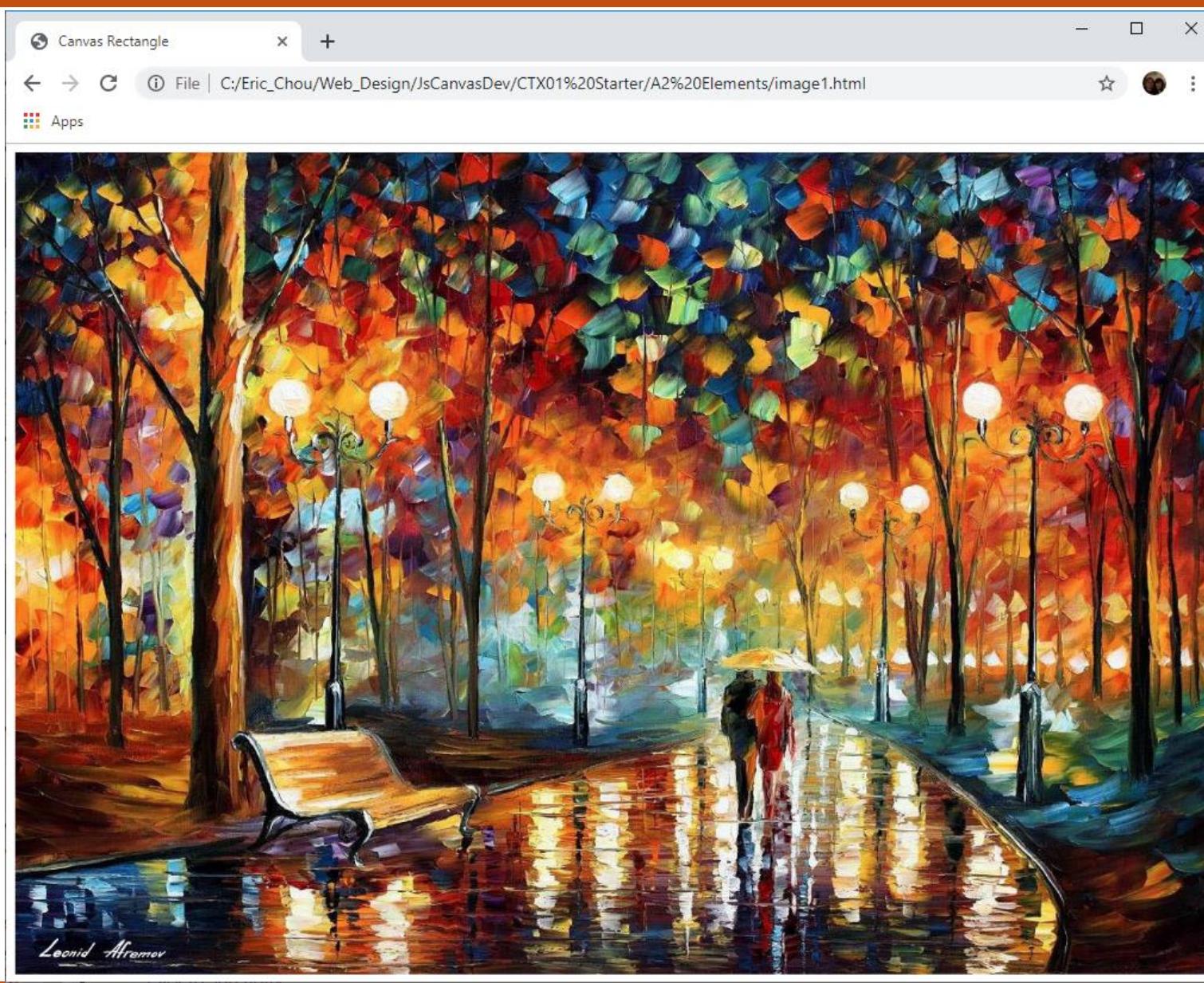
```
ctx.mozImageSmoothingEnabled = false;  
ctx.webkitImageSmoothingEnabled = false;  
ctx.msImageSmoothingEnabled = false;  
ctx.imageSmoothingEnabled = false;
```

```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11   </style>
12   <script type="javascript/text" src="init.js"></script>
13 </head>
14 ▼ <body>
15   <canvas id="canvas" width="1320px" height="1000px"></canvas>
16 ▼ <script>
17   var canvas = document.getElementById('canvas');
18   var ctx = canvas.getContext('2d');
19   var img = new Image();
20   img.src = "rains.jpg";
21 ▼   img.onload = function drawPattern() {
22     ctx.drawImage(img, 10, 10);
23   }
24 </script>
25 </body>
26 </html>
```



```
1 ▼ <html>
2 ▼   <head>
3     <title>Canvas Rectangle</title>
4 ▼   <style>
5     body{ margin:0; }
6 ▼   canvas{
7     width: 100%;
8     height: 100%;
9     background-color:white;
10    }
11  </style>
12  
13  <script type="javascript/text" src="init.js"></script>
14 </head>
15 ▼ <body>
16   <canvas id="canvas" width="1320px" height="1000px"></canvas>
17 ▼ <script>
18   var canvas = document.getElementById('canvas');
19   var ctx = canvas.getContext('2d');
20   var img = document.getElementById("rains");
21 ▼   window.onload = function(){
22     ctx.drawImage(img, 10, 10);
23   }
24 </script>
25 </body>
26 </html>
```





HTML5 Canvas Cheat Sheet v1.1

<http://blog.nihilogic.dk/>

Canvas element

Attributes

Name	Type	Default
width	unsigned long	300
height	unsigned long	150

Methods

Return	Name
string	toDataURL([Optional] string type, [Variadic] any args)
Object	getContext(string contextId)

2D Context

Attributes

Name	Type
canvas	HTMLCanvasObject [readonly]

Methods

Return	Name
void	save()
void	restore()

Transformation

Methods

Return	Name
void	scale(float x, float y)
void	rotate(float angle)
void	translate(float x, float y)
void	transform(float m11, float m12, float m21, float m22, float dx, float dy)
void	setTransform(float m11, float m12, float m21, float m22, float dx, float dy)

Image drawing

Methods

Return	Name
void	drawImage(Object image, float dx, float dy, [Optional] float dw, float dh)
void	drawImage(Object image, float sx, float sy, float sw, float sh, float dx, float dy, float dw, float dh)

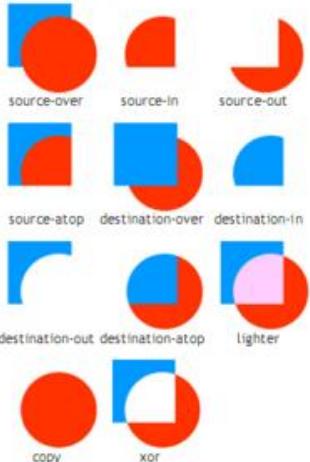
Argument "image" can be of type `HTMLImageElement`, `HTMLCanvasElement` or `HTMLVideoElement`

Compositing

Attributes

Name	Type	Default
globalAlpha	float	1.0
globalCompositeOperation	string	source-over

Supports any of the following values:



Line styles

Attributes

Name	Type	Default
lineWidth	float	1.0
lineCap	string	butt

Supports any of the following values:



lineJoin

Attributes

Supports any of the following values:



miterLimit

Attributes

Supports any of the following values:



Colors, styles and shadows

Attributes

Name	Type	Default
strokeStyle	any	black
fillStyle	any	black
shadowOffsetX	float	0.0
shadowOffsetY	float	0.0
shadowBlur	float	0.0
shadowColor	string	transparent black

Methods

Return	Name
CanvasGradient	createLinearGradient(float x0, float y0, float x1, float y1)
CanvasGradient	createRadialGradient(float x0, float y0, float r0, float x1, float y1, float r1)
CanvasPattern	createPattern(Object image, string repetition)

Argument "image" can be of type `HTMLImageElement`, `HTMLCanvasElement` or `HTMLVideoElement`

"repetition" supports any of the following values: [repeat (default), repeat-x, repeat-y, no-repeat]

CanvasGradient interface

void	addColorStop(float offset, string color)
------	--

CanvasPattern interface

No attributes or methods.

Paths

Methods

Return	Name
void	beginPath()
void	closePath()
void	fill()
void	stroke()
void	clip()
void	moveTo(float x, float y)
void	lineTo(float x, float y)
void	quadraticCurveTo(float cpx, float cpy, float x, float y)
void	bezierCurveTo(float cp1x, float cp1y, float cp2x, float cp2y, float x, float y)
void	arcTo(float x1, float y1, float x2, float y2, float radius)
void	arc(float x, float y, float radius, float startAngle, float endAngle, boolean anticlockwise)
void	rect(float x, float y, float w, float h)
boolean	isPointInPath(float x, float y)

Text

Attributes

Name	Type	Default
font	string	10px sans-serif
textAlign	string	start

Supports any of the following values: [start, end, left, right, center]

Methods

Return	Name
void	fillText(string text, float x, float y, [Optional] float maxWidth)
void	strokeText(string text, float x, float y, [Optional] float maxWidth)
TextMetrics	measureText(string text)

TextMetrics interface

width	float	[readonly]
-------	-------	------------

Rectangles

Methods

Return	Name
void	clearRect(float x, float y, float w, float h)
void	fillRect(float x, float y, float w, float h)
void	strokeRect(float x, float y, float w, float h)

Pixel manipulation

Methods

Return	Name
ImageData	createImageData(float sw, float sh)
ImageData	createImageData(ImageData)
ImageData	getImageData(float sx, float sy, float sw, float sh)
void	putImageData(ImageData, float dx, float dy, [Optional] float dirtyX, float dirtyY, float dirtyWidth, float dirtyHeight)

ImageData interface

width	unsigned long	[readonly]
height	unsigned long	[readonly]
data	CanvasPixelArray	[readonly]

CanvasPixelArray interface

length	unsigned long	[readonly]
--------	---------------	------------



Canvas, Audio and Video

LECTURE 6



Audio/Video/Graphics/Animation

This course includes:

- HTML5 Canvas
- Timing control for script-based animations
- HTML5 video and audio



Audio



<audio> tag and Audio() object

- Let's have something fun to start with. We are now talking about web browser supporting audio file in native, just like how tag is supported since 1994. HTML5 is likely to put an end to audio plugin such as Microsoft Windows Media player, Microsoft Silverlight, Apple QuickTime and the infamous Adobe Flash.



How to?

In order to make your web page plays music, the html code can be as simple as

```
<audio src="vincent.mp3" controls> </audio>
```

Unfortunately, the most popular audio format MPEG3(.mp3) is not an Open standard, it is patent encumbered. That means, web browser needs to pay a sum of money in order to decode it, and that might not be financially feasible for smaller company or organization. As you can see from table below, only those big boys are rich enough to decode MP3. Firefox and Opera supports only Vorbis (.ogg) format which is an Open standard.

On the other hand, the open standard Vorbis (*.ogg) is not supported by Safari and IE9. Hence, it is always good to have both Mp3 and Ogg side to side available.



Browser Support

Browser	.mp3	.wav	.ogg
Mozilla Firefox 3.6	✓	✓	
Opera 10.63		✓	✓
Google Chrome 8.0	✓	✓	✓
Apple Safari 5.0.3 (with QuickTime)	✓	✓	
Microsoft IE 9 Beta	✓	✓	



Demo Program: audio1.html

- Control Panel

```
<div>
<audio controls src="Raindrops.mp3"></audio>
</div>
```



▶ 0:00 / 2:11 ━ ━ ━



Demo Program: audio2.html

- Control the music play, pause, and reload by events.
- Hide the control panel.
- Write event-handlers for each audio playback functions.

```
1 ▼ <html>
2 ▼   <head>
3 ▼     <audio controls hidden id="music">
4       <source src="Raindrops.mp3" type="audio/mpeg">
5       Your browser does not support the audio element.
6     </audio>
7 ▼   <script>
8       var myMusic= document.getElementById("music");
9 ▼     function play() {
10       myMusic.play();
11     }
12
13 ▼     function pause() {
14       myMusic.pause();
15     }
16 ▼     function load() {
17       myMusic.load();
18     }
19 ▼     function ap(){
20       myMusic.play();
21     }
22       window.onload = ap;
23   </script>
24   </head>
25 ▼   <body onload="ap()">
26     <button onclick="play()" type="button">Play </button>
27     <button onclick="pause()" type="button">Pause</button>
28     <button onclick="load()" type="button">Reset</button>
29   </body>
30 </html>
```

Play

Pause

Reset

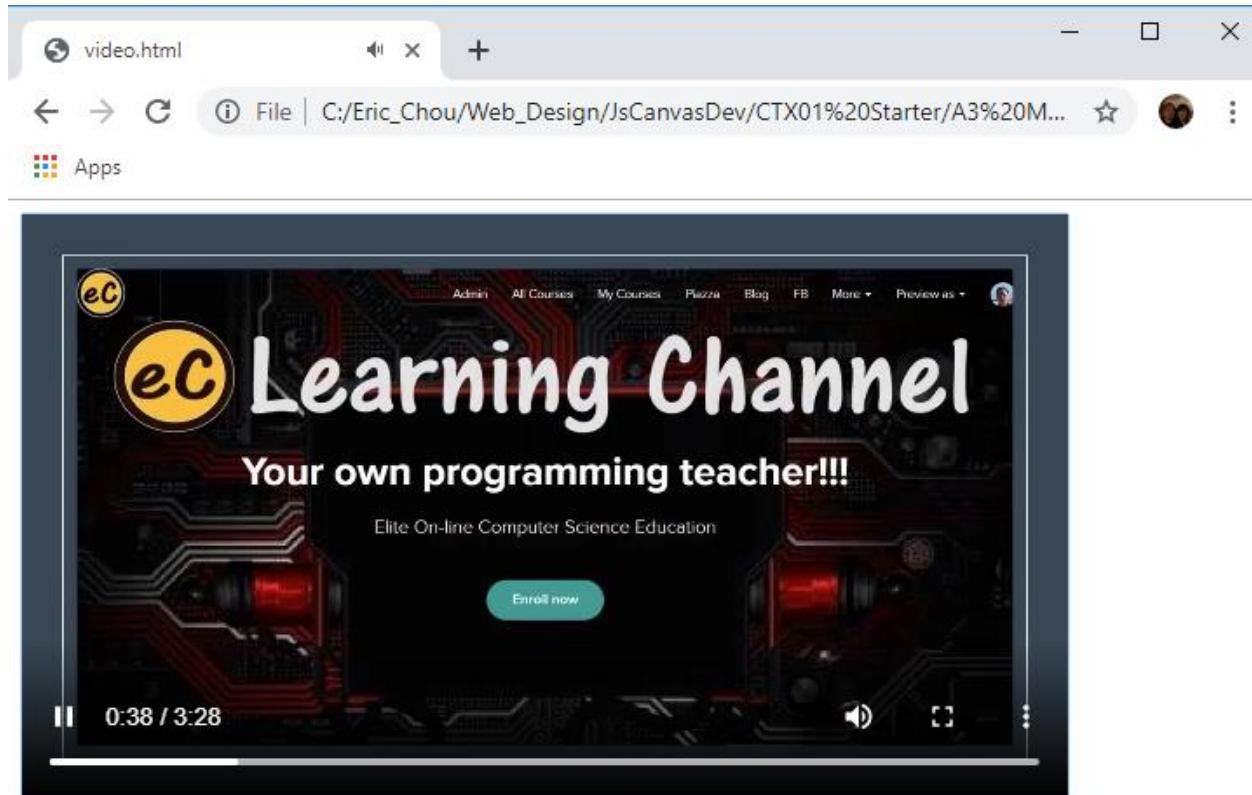


Video



Video

```
<html>
  <body>
    <video controls src="video.mp4">
      </video>
    </body>
</html>
```



Video

video.mp4

Animation

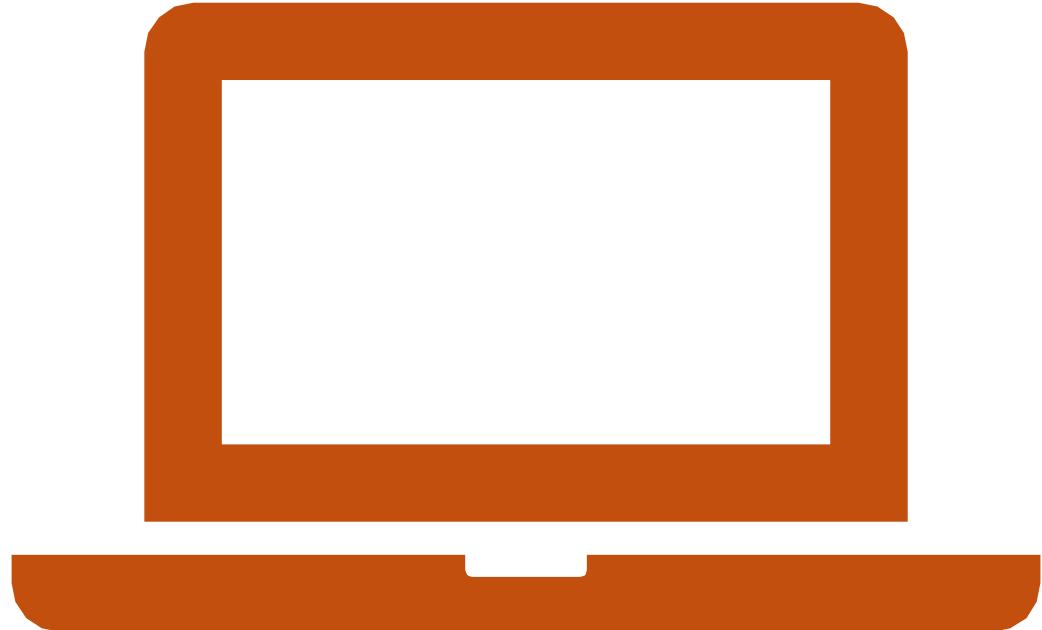




Simple Game Loop - setInterval

- `setInterval(Time_Up_Event_Handler, Time_in_milli_secs)`
- `Time_Up_Event_Handler` should be a function.
- `Time_in_milli_secs` should be an integer.





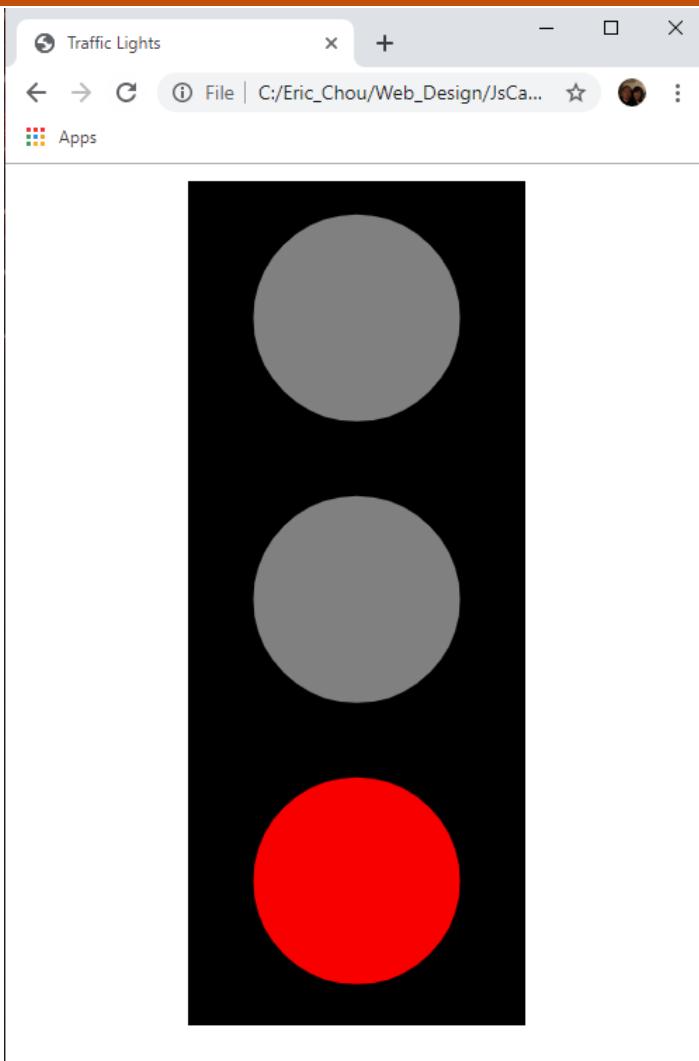
Demonstration Program

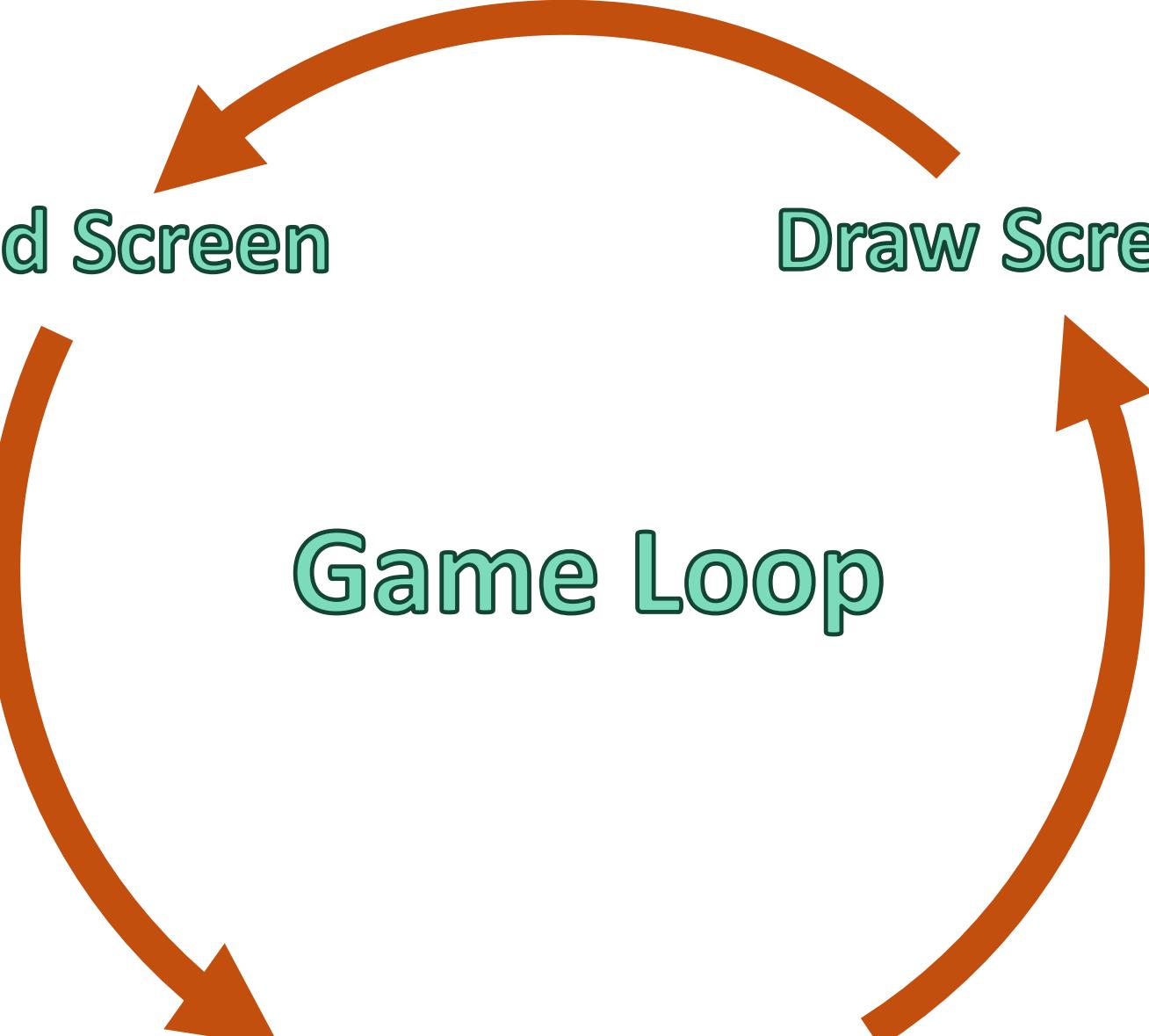
ANIME1.HTML + ANIME1.JS



Demo Program: anime1.html

- Simple 3-way no exit game loop.
- Draw Screen and Update Status in one handler function.
- Interval defined in setInterval();





Hold for an Interval

Hold Screen

In draw() function

Draw Screen

Game Loop

In draw() function

Update Status





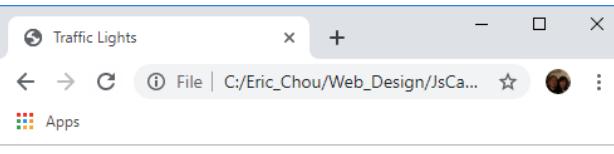
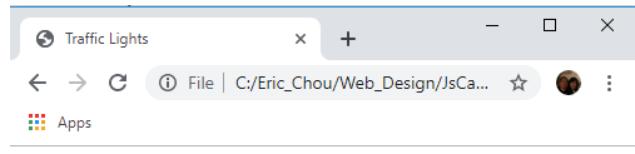
Simple Game Loop without Exit

- Design a state machine to control the game.
- Initial state to set up the initial condition
- Final state to set up the postgame condition.
- Provide an event to break the game loop.
- Provide an event to start the game loop.



Demo Program: anime2.html

- Start event: onclick of start button.
- Game over event: onclick of stop button.
- 4-state state machine.
- Both update status and draw screen in draw() function.





SVG and Canvas

LECTURE 6



Clarifying the SVG – Canvas Relationship

It's important to understand the differences between **SVG** and canvas elements. **SVG** is an **XML**-based vector graphics format. You can add styles to it with **CSS** and add dynamic behavior to it using the **SVG DOM**.

Canvas is bitmap based. It allows you to draw graphics and shapes through **JavaScript**. Like **SVG**, you may add style and dynamic behavior to it. Here are some reasons to use the canvas over **SVG**.

- When it comes to draw complex graphics, canvas is faster
- you can save images off the canvas whereas you can't using **SVG**
- everything in the canvas is a pixel.



Advantage of SVG

The SVG has some advantages too.

- Being resolutions independent, it can scale for different screen resolutions
- Since it is XML under the hood, targeting different elements is easier
- it's good at complex animations

So which one to choose over another? to develop a resolution dependent, highly interactive and vector-based graphics, choose SVG. If you to render graphics really fast, like in a game, or don't want to deal with XML, choose the canvas. Actually, they complement each other while delivering real-world applications.



Lab

WEB CLOCK



eC Learning Channel



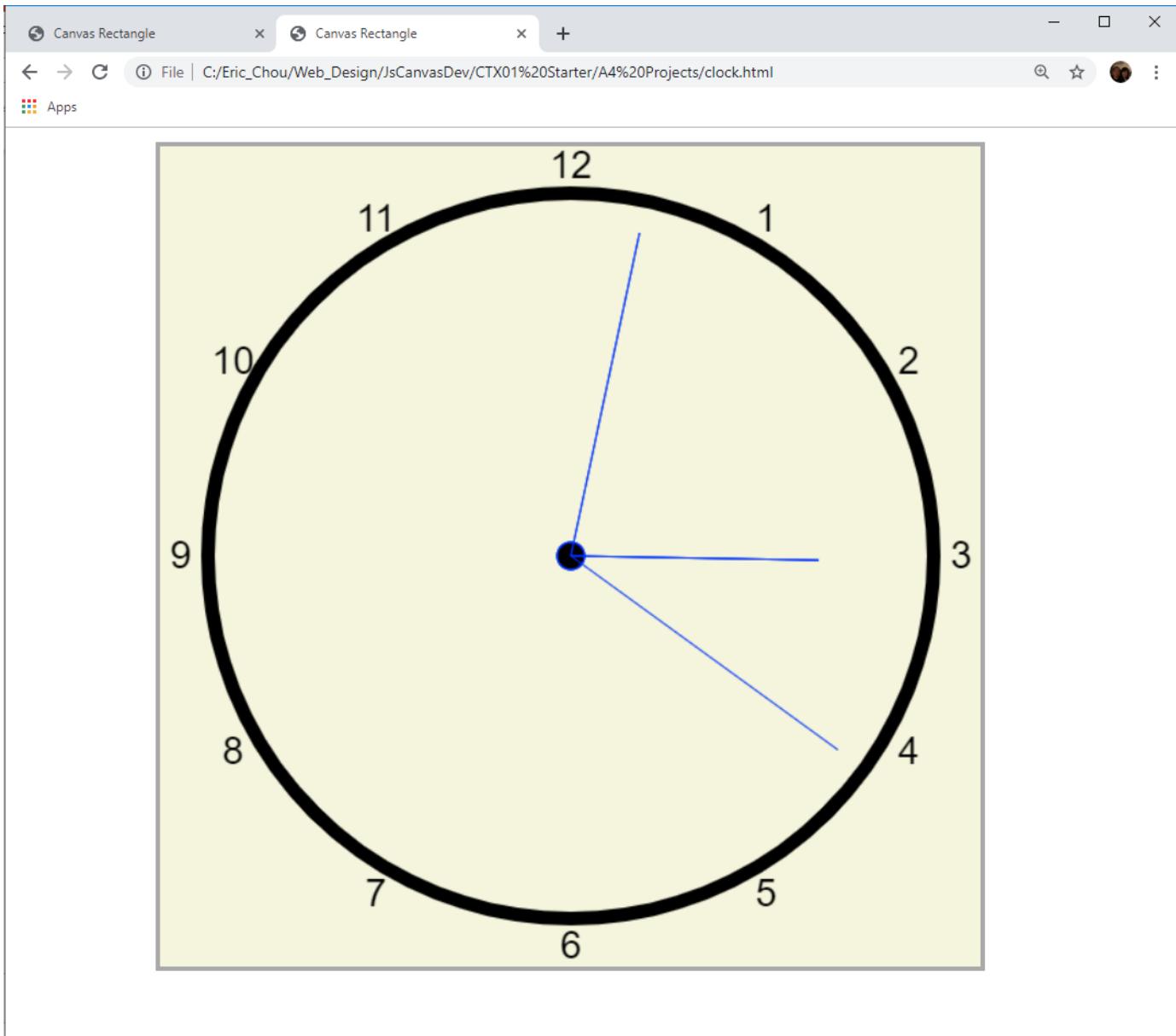
Demo Program: clock.html

- The clock application's **drawCircle()** method draws the circle representing the clock face by invoking **beginPath()** to begin a path, and subsequently invokes **arc()** to create a circular path. That path is invisible until the application invokes **stroke()**. Likewise, the application's **drawCenter()** method draws the small filled circle at the center of the clock with a combination of **beginPath()**, **arc()**, and **fill()**.
- The application's **drawNumerals()** method draws the numbers around the face of the clock with the **fillText()** method, which draws filled text in the canvas.
- Unlike the **arc()** method, **fillText()** does not create a path; instead, **fillText()** immediately renders text in the canvas.



Demo Program: clock.html

- The clock hands are drawn by the application's **drawHand()** method, which uses three methods to draw the lines that represent the clock hands: **moveTo()**, **lineTo()**, and **stroke()**. The **moveTo()** method moves the graphics pen to a specific location in the canvas, **lineTo()** draws an invisible path to the location that you specify, and **stroke()** makes the current path visible.
- The application animates the clock with **setInterval()**, which invokes the application's **drawClock()** function once every second. The **drawClock()** function uses **clearRect()** to erase the canvas, and then it redraws the clock.



HTML



CANVAS