CS 50 Web Design

APCSP Module 2: Internet

Unit 4: Web Graphics Design

LECTURE 14: BASIC CANVAS GRAPHICS DESIGN

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- Learn what is Canvas?
- How to use it?
- Learn how to setup Canvas frame?
- Learn how to create Canvas graphics?

Introduction

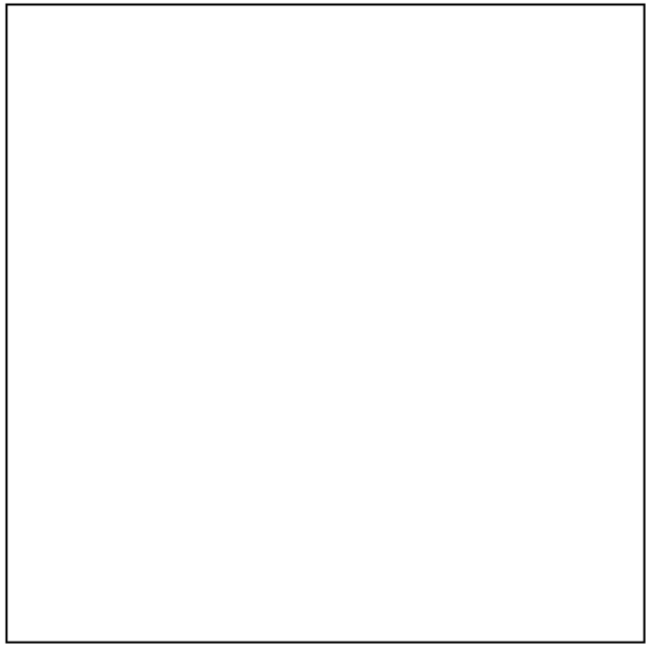
SECTION 1

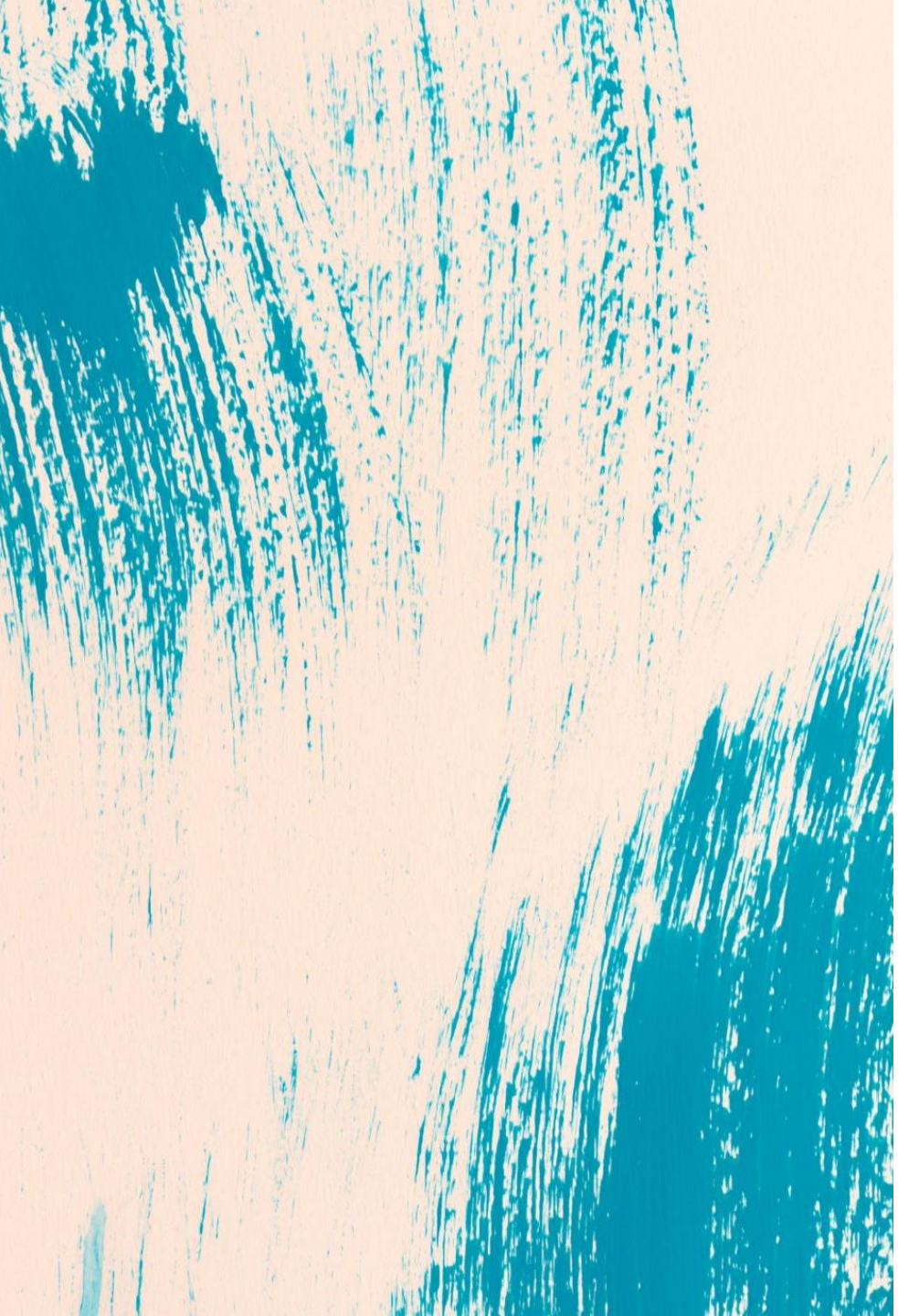
Overview

- 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?

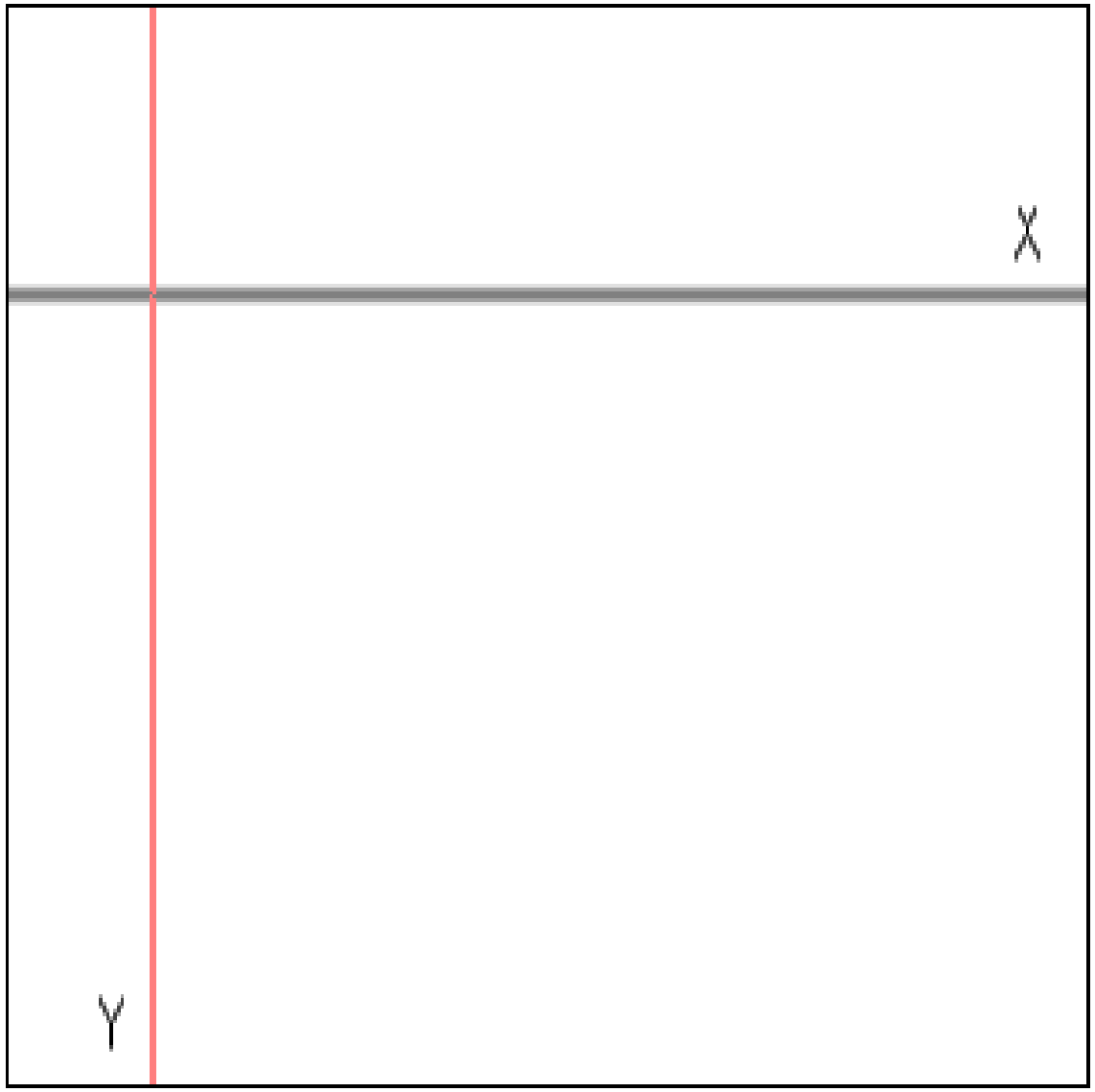
- Officially a canvas is "a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly".
- Simply saying, with the help of **JavaScript** and **HTML5 canvas** element you may render 2D shapes and bitmap images. The image below shows the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

Canvas Overview

SECTION 1

Demonstration Program

STARTER.HTML

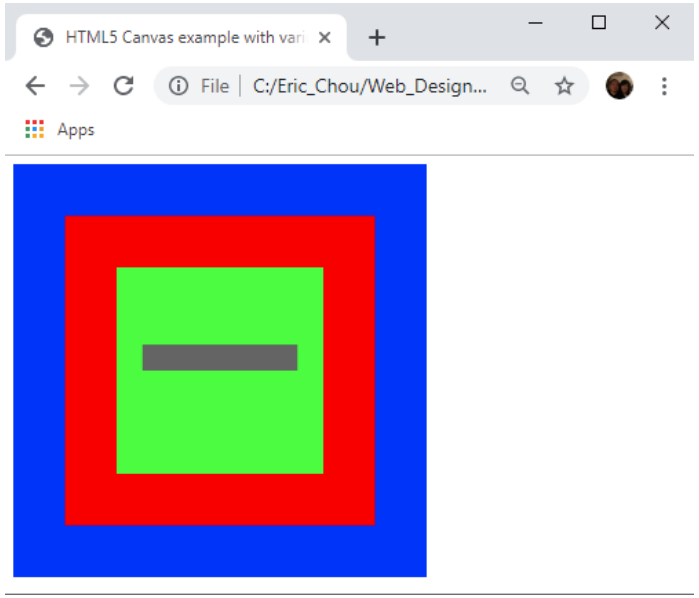
```html
<!DOCTYPE html>
<html>
    <head>
        <title>HTML5 Canvas Demo</title>
        <style>
            #FirstCanvas{
            width: 500px;
            height: 300px;
            border: 3px solid green;
            background-color: orange;
            }
        </style>
    </head>
    <body>
        <canvas id="FirstCanvas"></canvas>
    </body>
</html>

```

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

```
<html>
    <head>
    <title>HTML5 Canvas example with various methods </title>
        <script>
         window.onload=function() {
           var canvas=document.getElementById("canvas");
           var ctx=canvas.getContext('2d');
               ctx.fillStyle='rgb(0,0,255)';
               ctx.fillRect(0,0,400,400);
               ctx.fillStyle='rgb(255,0,0)';
               ctx.fillRect(50,50,300,300);
               ctx.fillStyle='rgb(0,255,0)';
               ctx.fillRect(100,100,200,200);
               ctx.fillStyle='rgb(100,100,100)';
               ctx.fillRect(125,175,150,25);
         }
        </script>
    </head>
    <body>
        <div>
            <canvas id="canvas" width="400" height="400"></canvas>
        </div>
    </body>
</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

SECTION 1

Setting Up Canvas

- To set up a canvas for drawing, your 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 ("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.

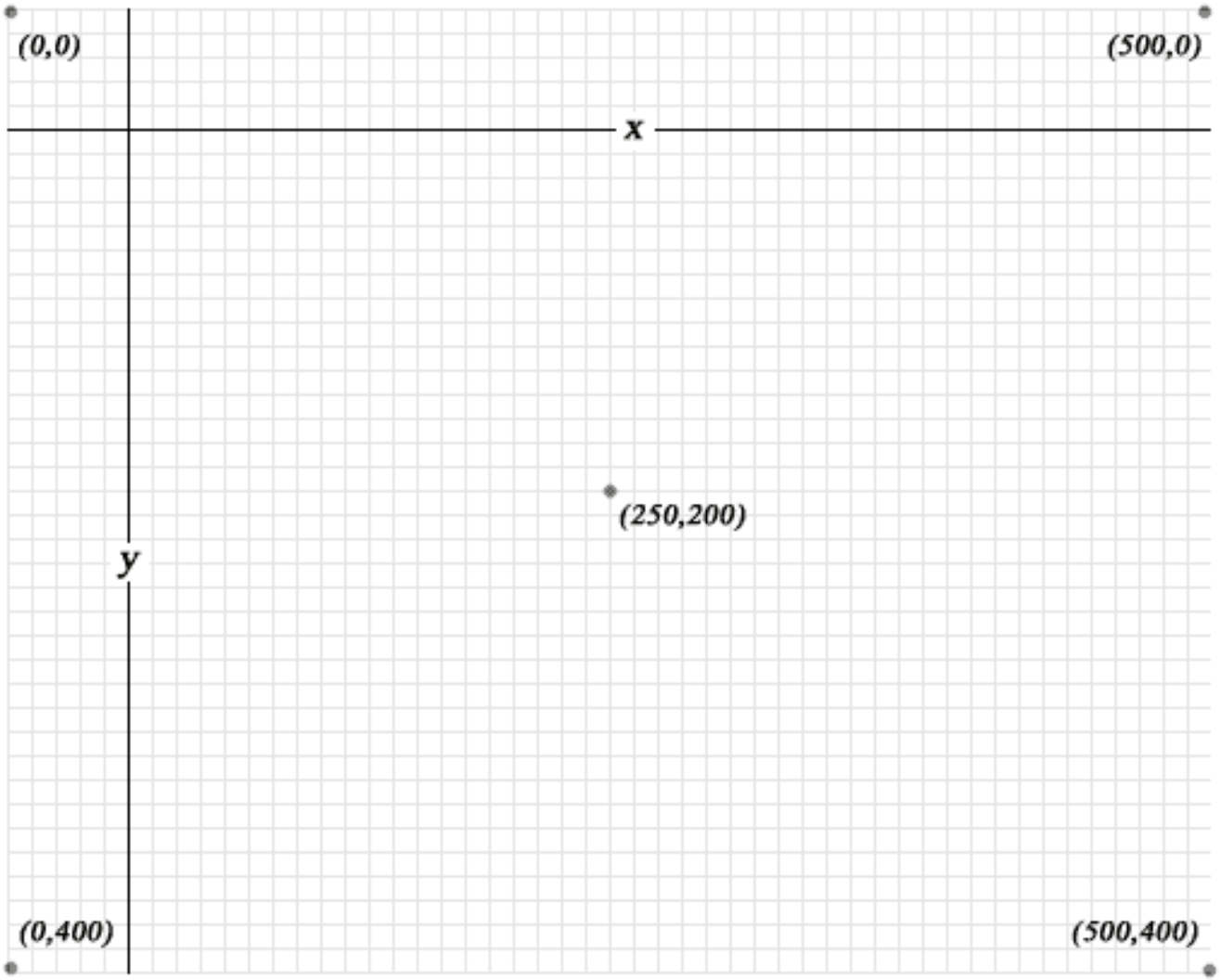
Checking for support

- 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) coinsides that of canvas's.

Demonstration Program

STARTER3.HTML

```
<html>
    <head>
        <meta charset=utf-8 />
        <title>Draw a line</title>
        <style>canvas{ background-color: beige}</style>
    </head>
    <body>
        <canvas id="canvas" width="500" height="400"></canvas>
        <script>
            var canvas = document.getElementById('canvas');
            //Always check for properties and methods, to make sure your
            code doesn't break in other browsers.
            if (canvas.getContext)
            {
             var ctx = canvas.getContext('2d');
                 // drawing code here
                 alert("My Browser support Canvase");
            }
            else
            {
                // canvas-unsupported code here
                 alert("My Browser doesn't support Canvase");
            }
        </script>
    </body>
</html>

```



Pictorial Presentation of the above Canvas :

Full Page Canvas

SECTION 1

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')
```

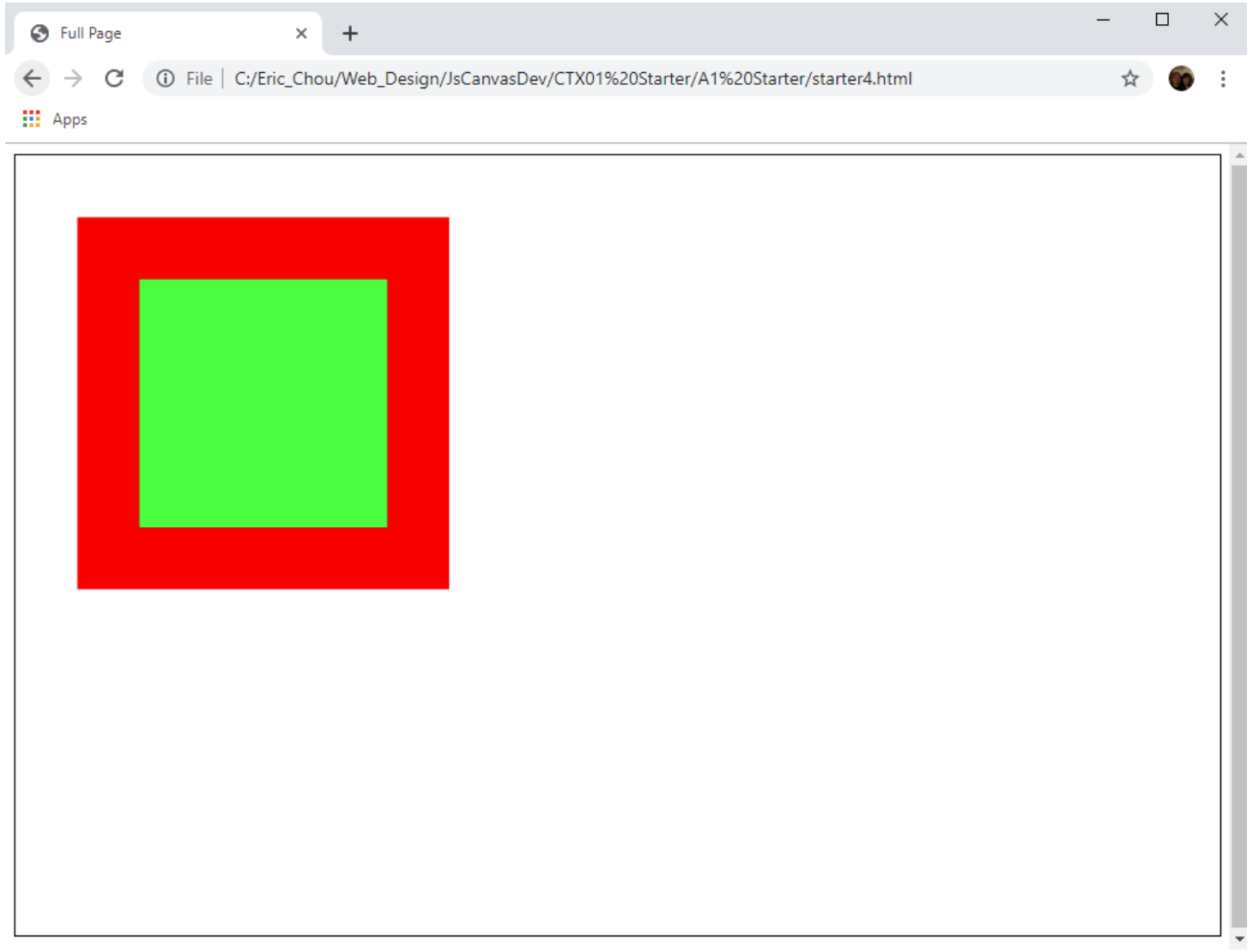
Demonstration Program

STARTER4.HTML

```
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>
```

Demo Program: starter4.html

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

Debouncing

SECTION 1

Demonstration Program

STARTER5.HTML

Demonstration Program

STARTER5.HTML

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):

```
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

External JavaScript File

SECTION 1

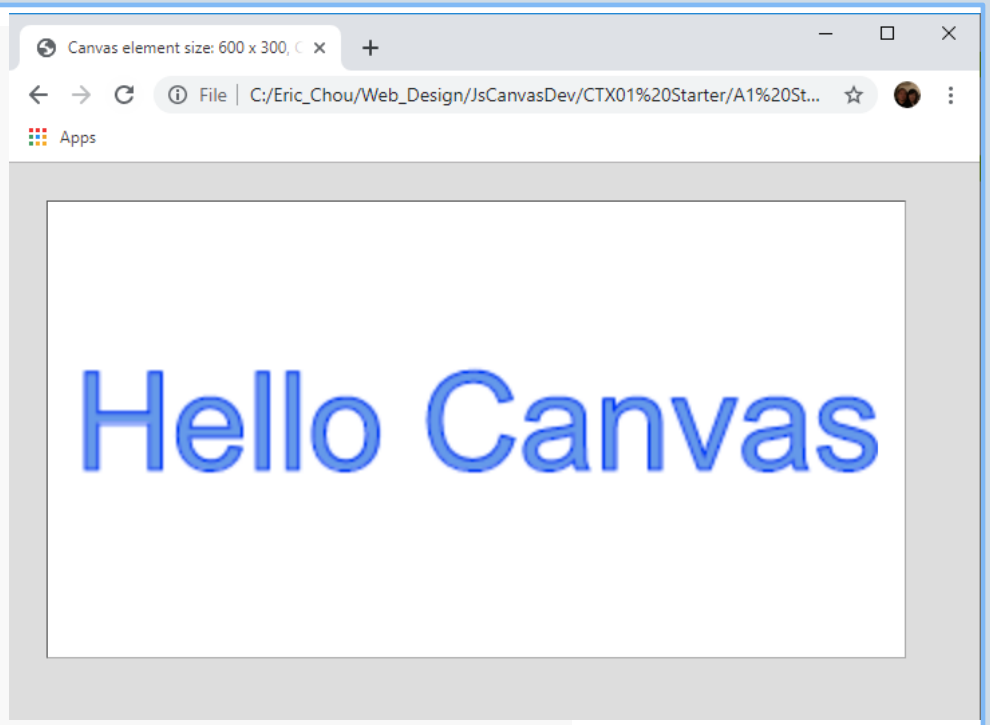
File Structure

- 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

```html
<!DOCTYPE html>
<html>
<head>
    <title>Canvas element size: 600 x 300,
    Canvas drawing surface size: 300 x 150</title>
    <style>
        body {
            background: #dddddd;
        }
        #canvas {
            margin: 20px;
            padding: 20px;
            background: #ffffff;
            border: thin inset #aaaaaa;
            width: 600px;
            height: 300px;
        }
    </style>
    </head>
    <body>
        <canvas id='canvas'>
            Canvas not supported
        </canvas>
        <script src='starter6.js'></script>
    </body>
</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 );
```

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

SECTION 1

Draw elements to a canvas

With the context we can now draw elements.

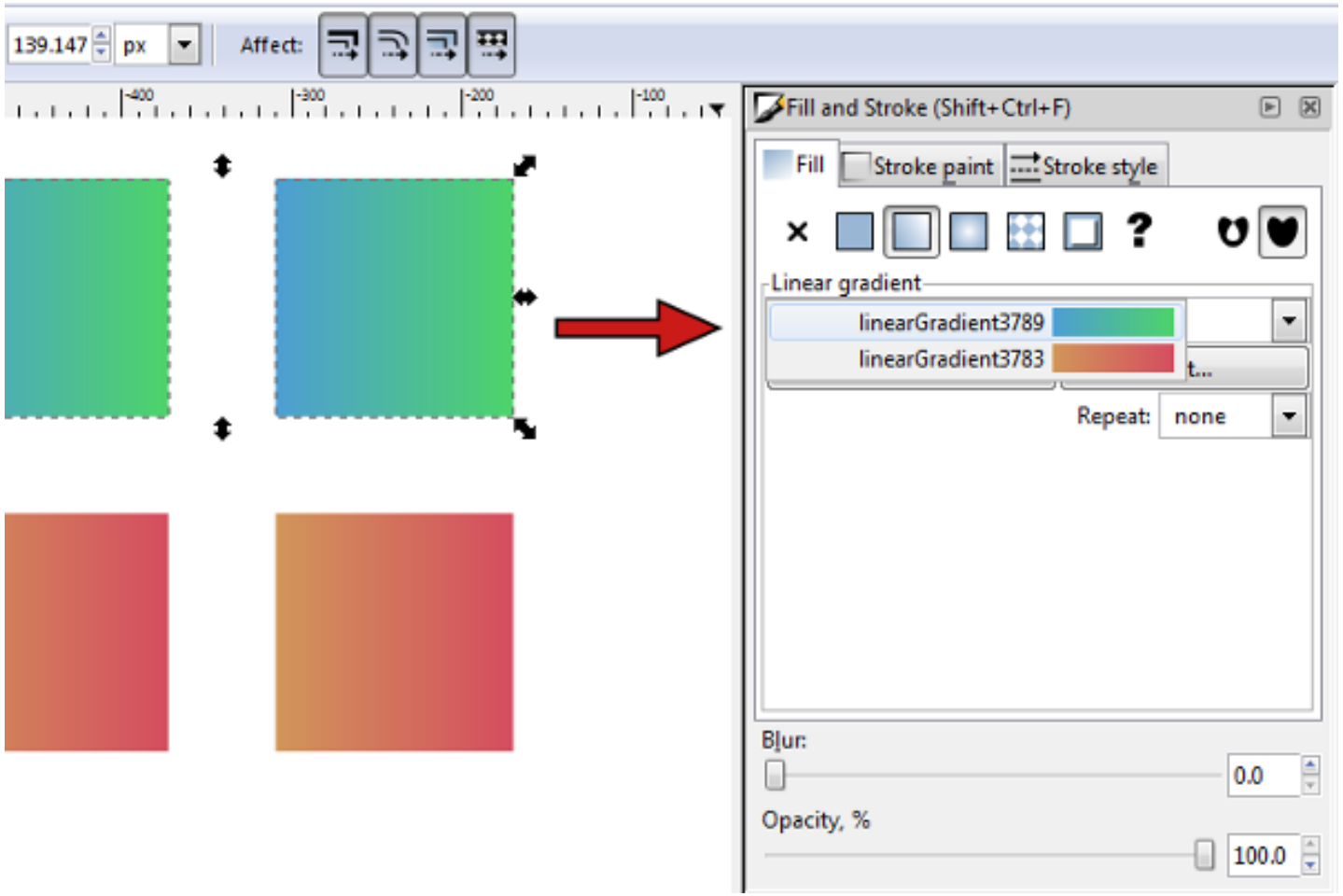
We have several methods to do so. We can draw:

- 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:

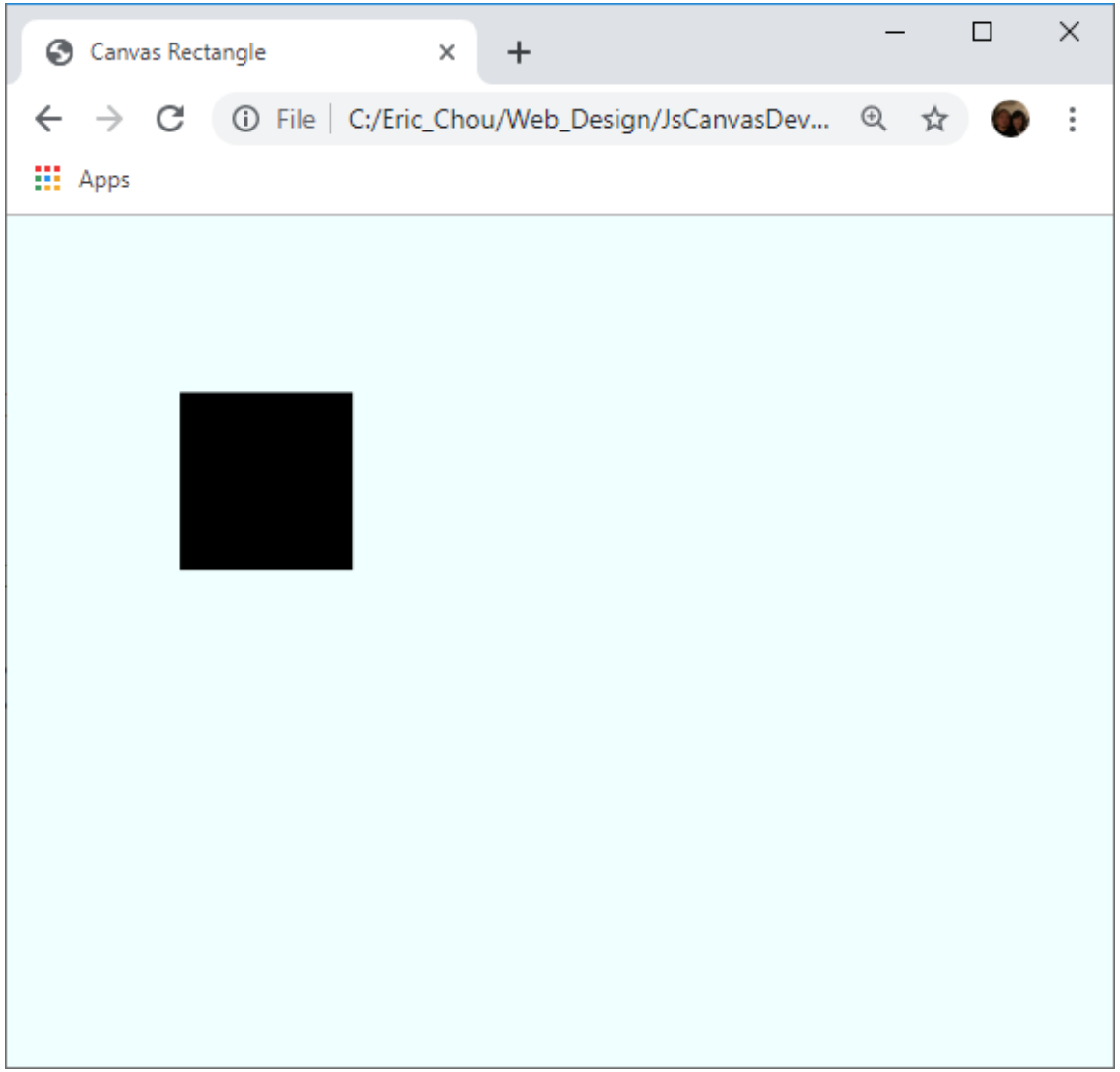
```
c.fillStyle = 'white'
c.fillRect(100, 100, 100, 100)
```

rectangle.html

```
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:azure;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.querySelector('canvas');
            var ctx= canvas.getContext('2d');

            ctx.fillStyle='black';
            ctx.fillRect(100,100,100,100);
        </script>
    </body>
</html>
```

init.js

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

canvas.width  = window.innerWidth;
canvas.height = window.innerHeight;
if (canvas.getContext)
    {   ctx = canvas.getContext('2d');
    }

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;
}))
```


Canvas Rectangle
File
C:/Eric_Chou/Web_Design/JsCanvasDev...
Apps

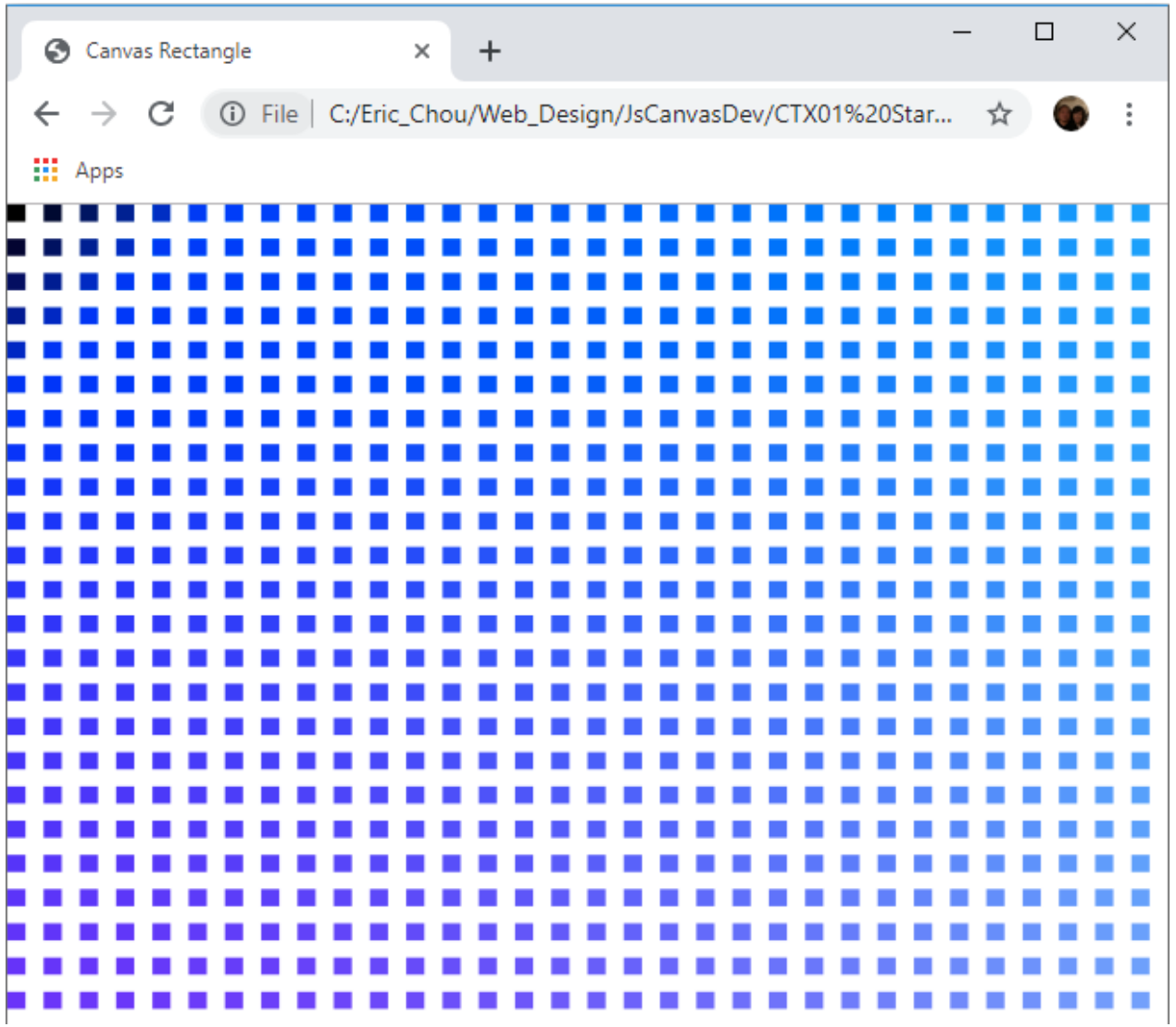
Demo Program: rectangle2.html+init.js

- 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)
  }
}
```

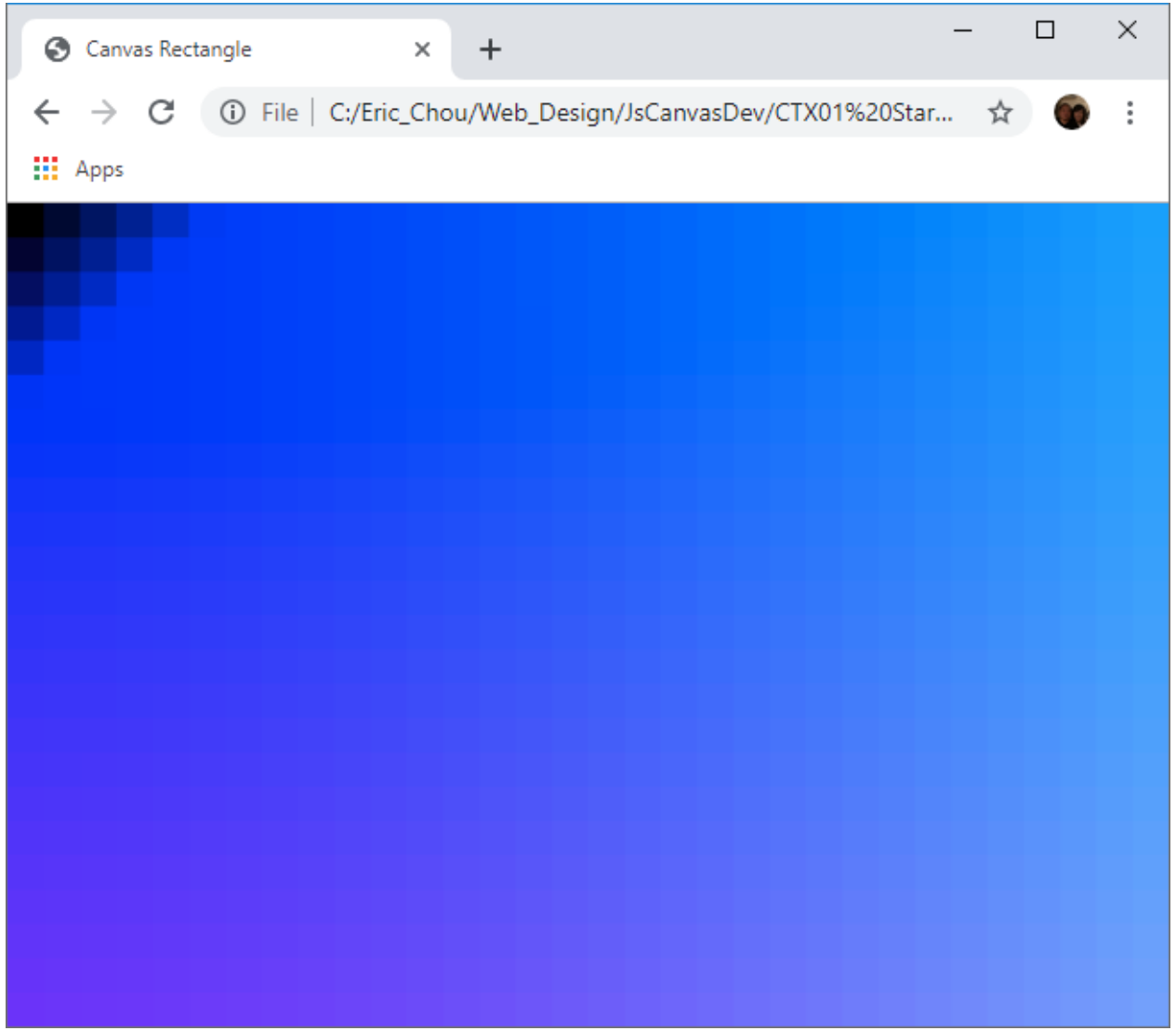
rectangle2.html

```
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.querySelector('canvas');
            var ctx= canvas.getContext('2d');

            for (let i = 0; i < 60; i++) {
              for (let j = 0; j < 60; j++) {
                ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`
                ctx.fillRect(j * 20, i * 20, 10, 10)
              }
            }
        </script>
    </body>
</html>
```


Canvas Rectangle
File
C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Star...
Apps

rectangle3.html

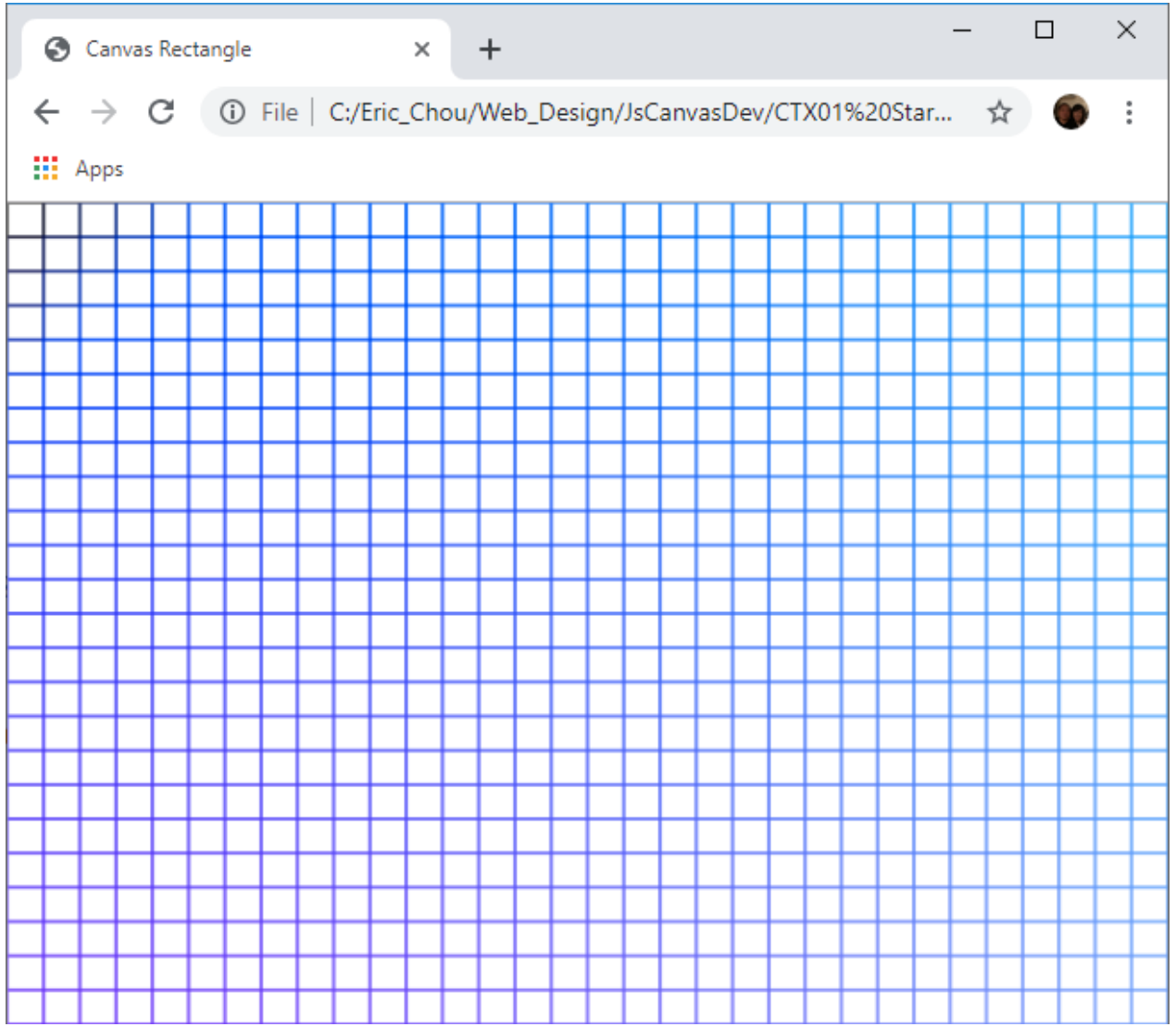
```html
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.querySelector('canvas');
            var ctx= canvas.getContext('2d');

            for (let i = 0; i < 60; i++) {
              for (let j = 0; j < 60; j++) {
                ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`
                ctx.fillRect(j * 20, i * 20, 20, 20)
              }
            }
        </script>
    </body>
</html>
```


Canvas Rectangle
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Star...
Apps

```html
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.querySelector('canvas');
            var ctx= canvas.getContext('2d');

            for (let i = 0; i < 60; i++) {
              for (let j = 0; j < 60; j++) {
                ctx.strokeStyle = `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`
                ctx.strokeRect(j * 20, i * 20, 20, 20)
              }
            }
        </script>
    </body>
</html>
```

rectangle4.html

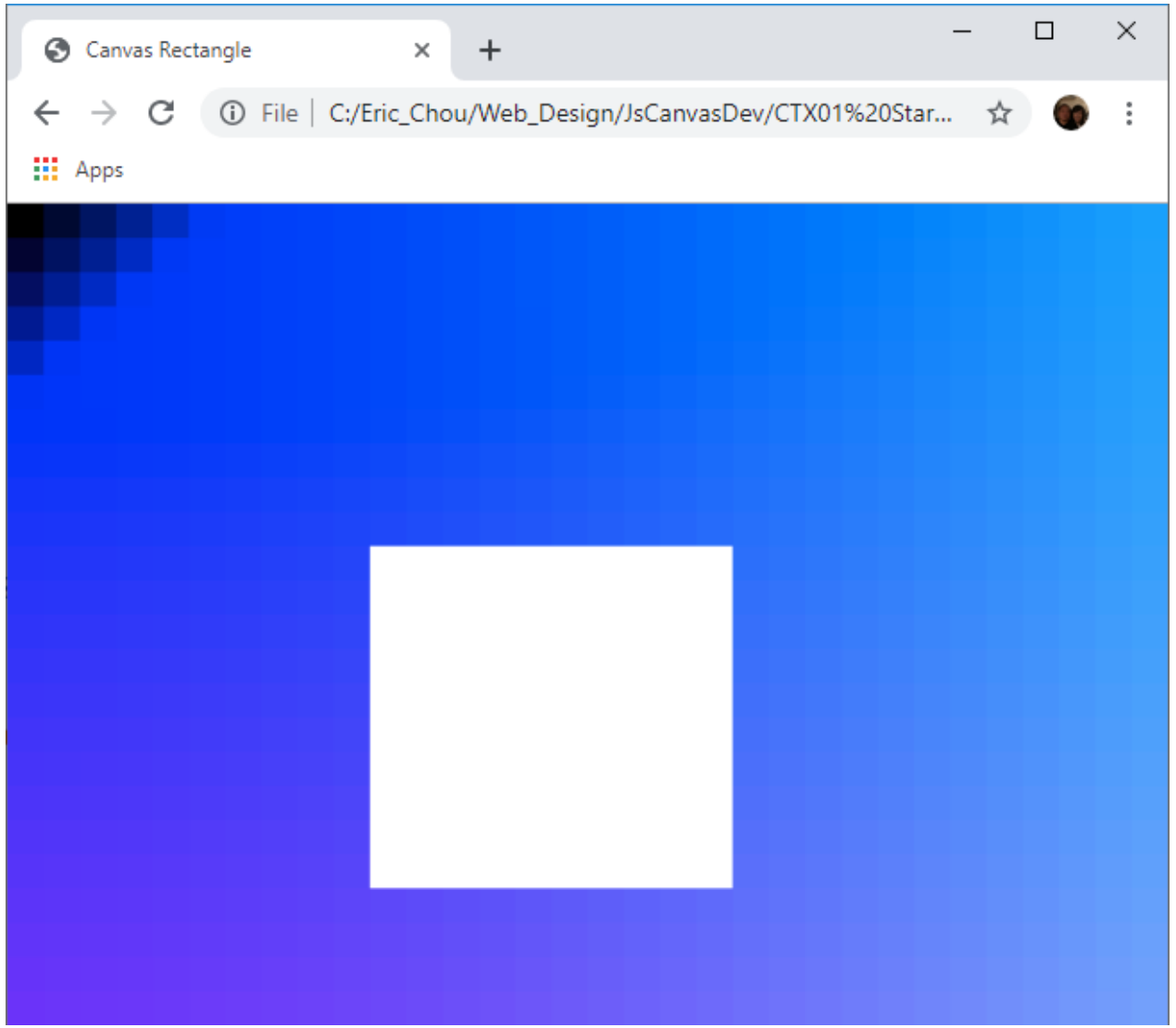
```
ctx.strokeStyle =
  `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`

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


Canvas Rectangle
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Star...
Apps

rectangle5.html

clearRect() sets an area as transparent:

```
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.querySelector('canvas');
            var ctx= canvas.getContext('2d');

            for (let i = 0; i < 60; i++) {
              for (let j = 0; j < 60; j++) {
                ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`
                ctx.fillRect(j * 20, i * 20, 20, 20)
              }
            }
            ctx.clearRect(200, 200, 200, 200)
        </script>
    </body>
</html>
```

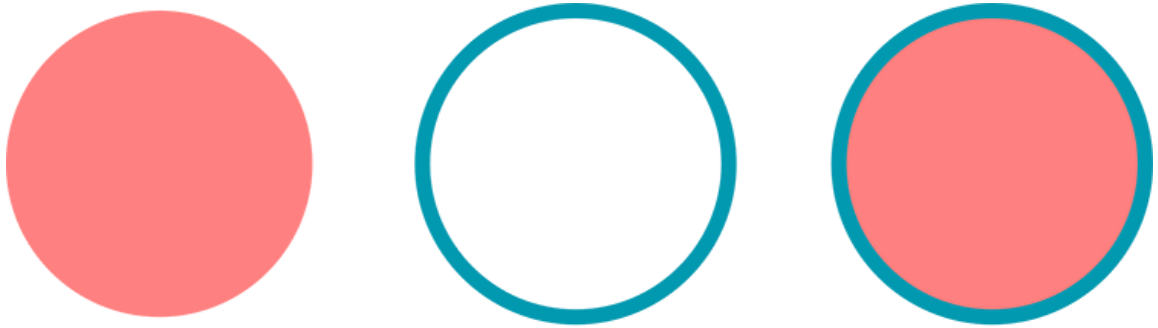

Canvas Rectangle
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Star...
Apps

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):

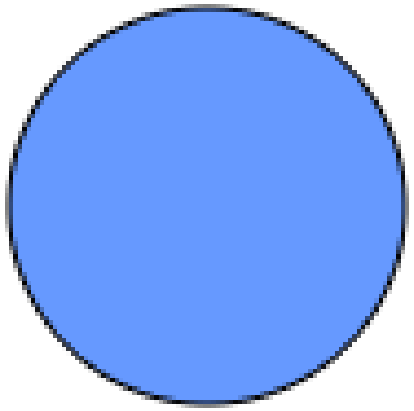
```
const c = canvas.getContext('2d')
for (let i = 0; i < 61; i++) {
  for (let j = 0; j < 61; j++) {
    c.strokeStyle = `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`
    c.strokeRect(j * 20, i * 20, 20, 20)
  }
}
```


HTML-5 Canvas
Shapes

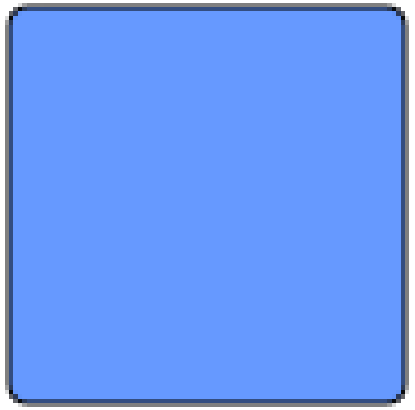
12 Shapes to Extend HTML5 Canvas

ctx.fillArea(x, y, color)

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

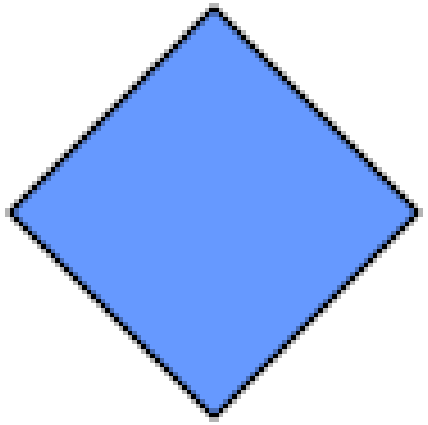


ctx.roundedRect(x, y, w, h, radius)

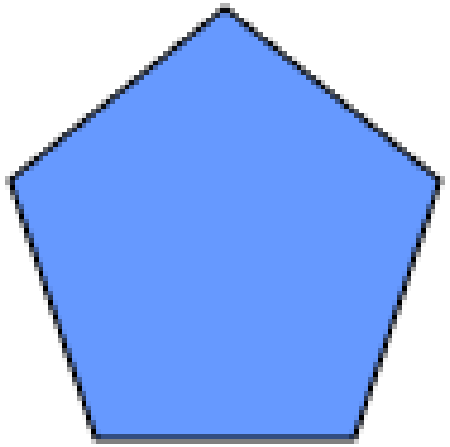


12 Shapes to Extend HTML5 Canvas

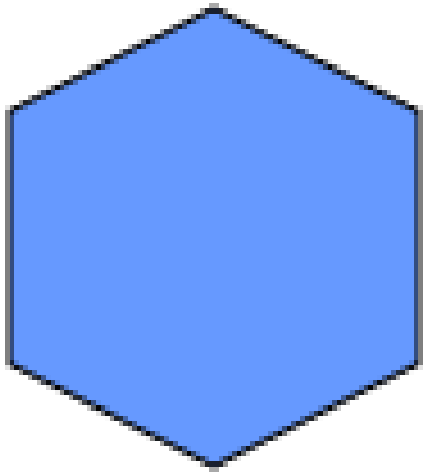
ctx.diamond(x, y, w, h)



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

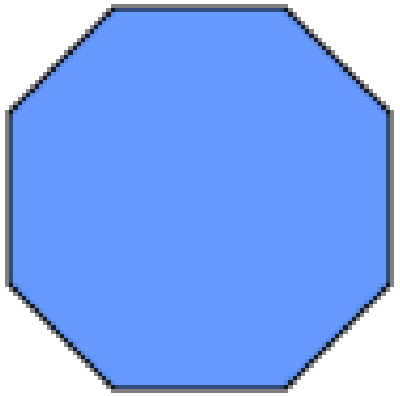


ctx.hexagon(x, y, w, h)



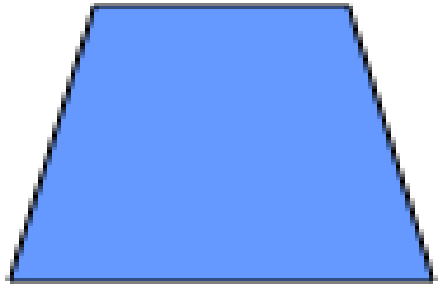
12 Shapes to Extend HTML5 Canvas

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



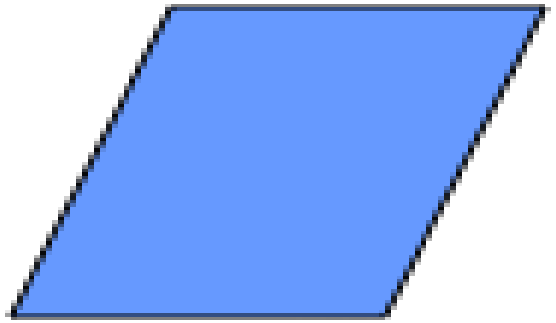
ctx.star(x, y, w, h)

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

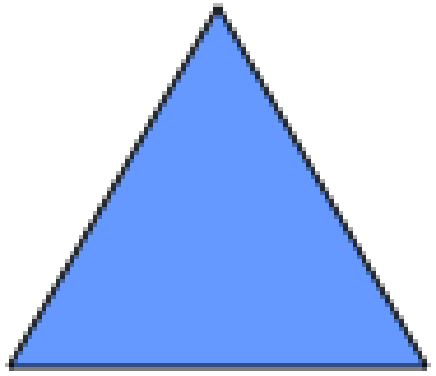


12 Shapes to Extend HTML5 Canvas

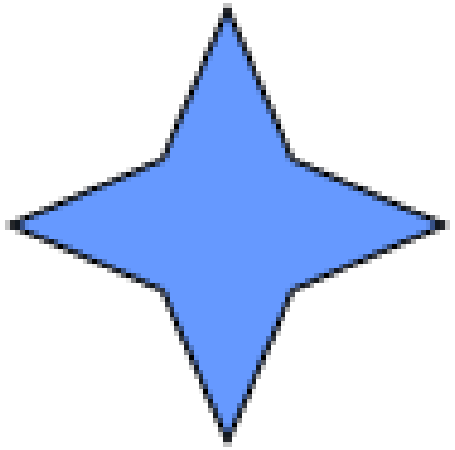
ctx.parallelogram(x, y, w, h)



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



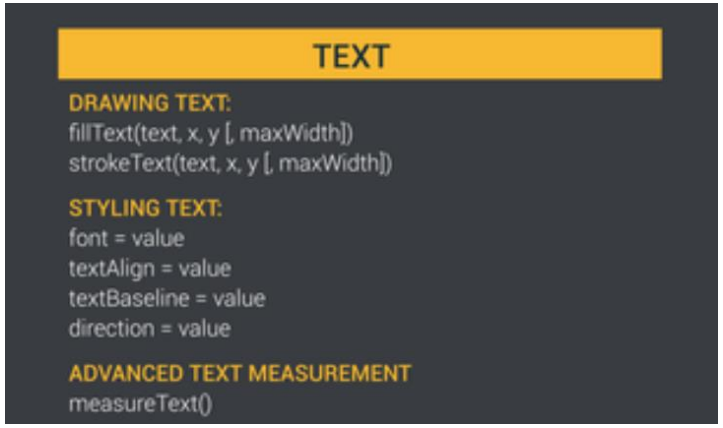
ctx.ninjaStar(x, y, w, h)



Text

ACTIVITY

Text



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:

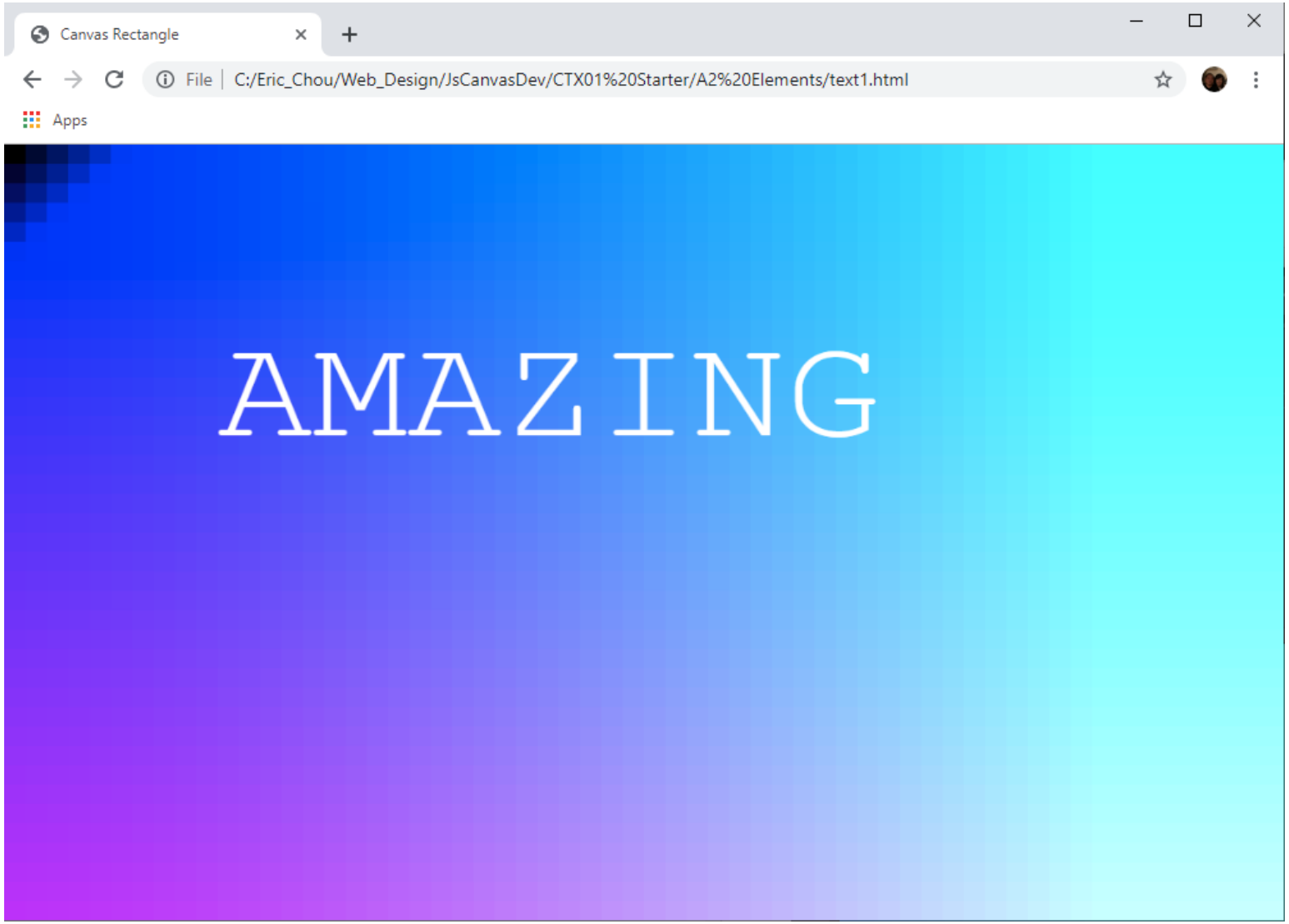
```
c.font = '148px Courier New'
```

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*)

text1.html

```
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="1200px" height="800px"></canvas>
        <script>
            var canvas = document.querySelector('canvas');
            var ctx= canvas.getContext('2d');

            for (let i = 0; i < 60; i++) {
              for (let j = 0; j < 60; j++) {
                ctx.fillStyle = `rgb(${i * 5}, ${j * 5}, ${(i+j) * 50})`
                ctx.fillRect(j * 20, i * 20, 20, 20)
              }
            }
            ctx.font = '148px Courier New';
            ctx.fillText('AMAZING', 200, 300);
        </script>
    </body>
</html>
```


Canvas Rectangle
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Starter/A2%20Elements/text1.html
Apps
AMAZING

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.

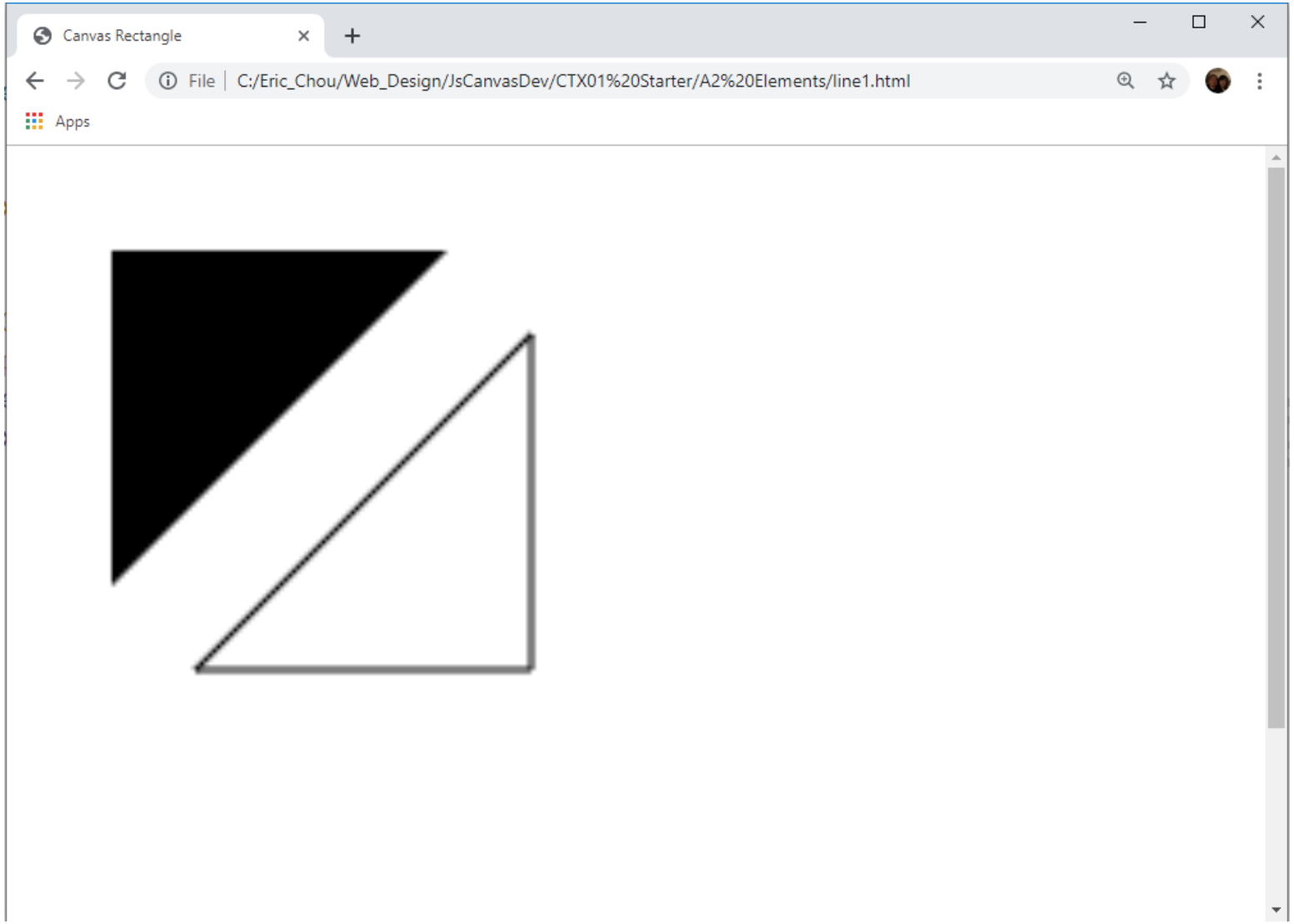
Polygon by Line

Filled triangle

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

line1.html

```
<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>
```


Canvas Rectangle
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Starter/A2%20Elements/line1.html
Apps

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();
```

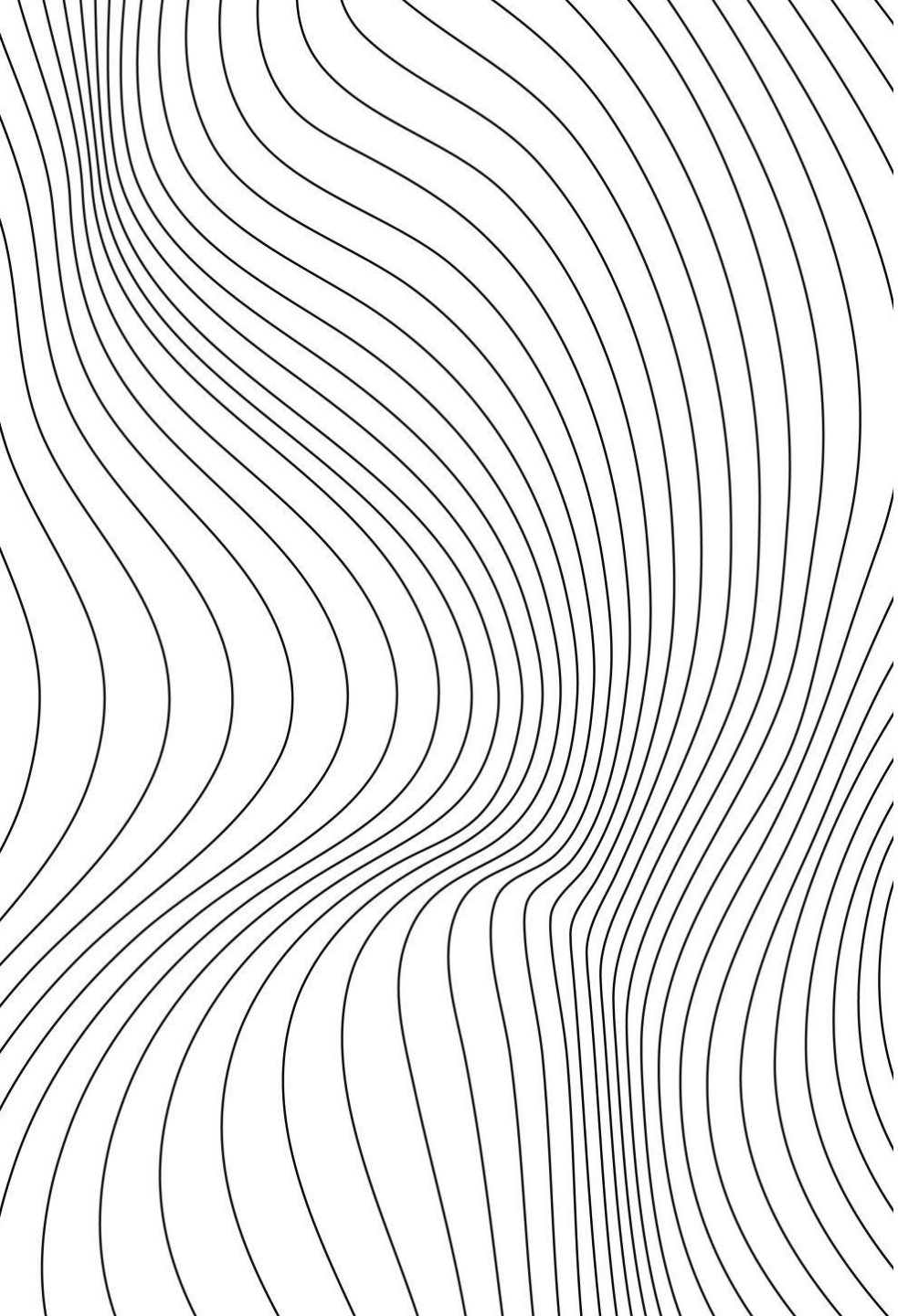
- Every time you call it, is like you place your pen above the canvas and you are about to write.
- Also initially the pen is not positioned anywhere....is "in the air"

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)

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

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

Green colors

GreenYellow	AD FF 2F	173 255 47
Chartreuse	7F FF 00	127 255 0
LawnGreen	7C FC 00	124 252 0
Lime	00 FF 00	0 255 0
LimeGreen	32 CD 32	50 205 50
PaleGreen	98 FB 98	152 251 152
LightGreen	90 EE 90	144 238 144
MediumSpringGreen	00 FA 9A	0 250 154
SpringGreen	00 FF 7F	0 255 127
MediumSeaGreen	3C B3 71	60 179 113
SeaGreen	2E 8B 57	46 139 87
ForestGreen	22 8B 22	34 139 34
Green	00 80 00	0 128 0
DarkGreen	00 64 00	0 100 0
YellowGreen	9A CD 32	154 205 50
OliveDrab	6B 8E 23	107 142 35
Olive	80 80 00	128 128 0
DarkOliveGreen	55 6B 2F	85 107 47
MediumAquamarine	66 CD AA	102 205 170
DarkSeaGreen	8F BC 8F	143 188 143
LightSeaGreen	20 B2 AA	32 178 170
DarkCyan	00 8B 8B	0 139 139
Teal	00 80 80	0 128 128

Blue/Cyan colors

Aqua	00 FF FF	0 255 255
Cyan	00 FF FF	0 255 255
LightCyan	E0 FF FF	224 255 255
PaleTurquoise	AF EE EE	175 238 238
Aquamarine	7F FF D4	127 255 212
Turquoise	40 E0 D0	64 224 208
MediumTurquoise	48 D1 CC	72 209 204
DarkTurquoise	00 CE D1	0 206 209
CadetBlue	5F 9E A0	95 158 160
SteelBlue	46 82 B4	70 130 180
LightSteelBlue	B0 C4 DE	176 196 222
PowderBlue	B0 E0 E6	176 224 230
LightBlue	AD D8 E6	173 216 230
SkyBlue	87 CE EB	135 206 235
LightSkyBlue	87 CE FA	135 206 250
DeepSkyBlue	00 BF FF	0 191 255
DodgerBlue	1E 90 FF	30 144 255
CornflowerBlue	64 95 ED	100 149 237
MediumSlateBlue	7B 68 EE	123 104 238
RoyalBlue	41 69 E1	65 105 225
MediumBlue	00 00 CD	0 0 205
DarkBlue	00 00 8B	0 0 139
Navy	00 00 80	0 0 128
MidnightBlue	19 19 70	25 25 112

Brown colors

Cornsilk	FF F8 DC	255 248 220
BlanchedAlmond	FF EB CD	255 235 205
Bisque	FF E4 C4	255 228 196
NavajoWhite	FF DE AD	255 222 173
Wheat	F5 DE B3	245 222 179
BurlyWood	DE B8 87	222 184 135
Tan	D2 B4 8C	210 180 140
RosyBrown	BC 8F 8F	188 143 143
SandyBrown	F4 A4 60	244 164 96
Goldenrod	DA A5 20	218 165 32
DarkGoldenrod	B8 86 0B	184 134 11
Peru	CD 85 3F	205 133 63
Chocolate	D2 69 1E	210 105 30
SaddleBrown	8B 45 13	139 69 19
Sienna	A0 52 2D	160 82 45
Brown	A5 2A 2A	165 42 42
Maroon	80 00 00	128 0 0

White colors

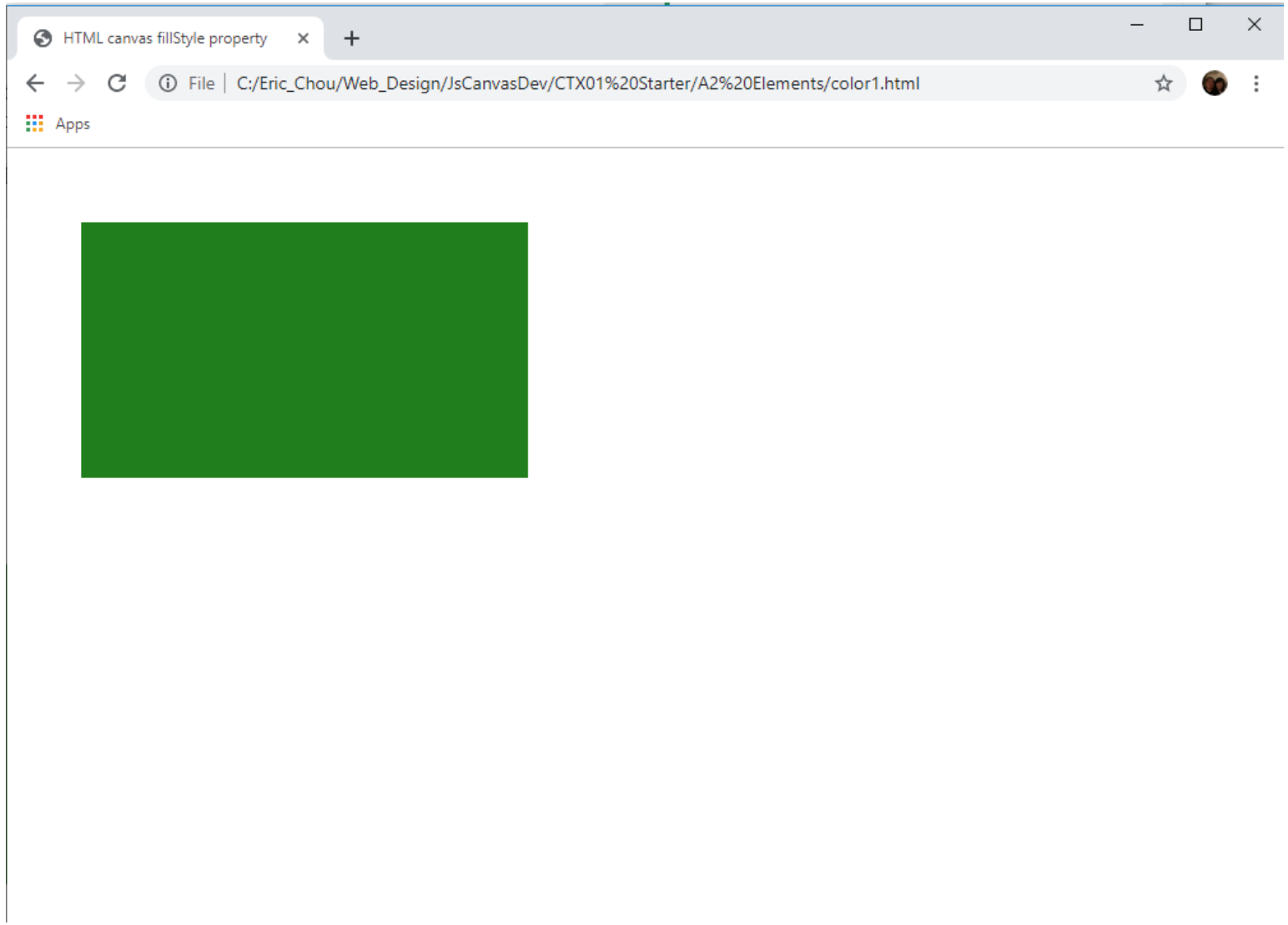
White	FF FF FF	255 255 255
Snow	FF FA FA	255 250 250
Honeydew	F0 FF F0	240 255 240
MintCream	F5 FF FA	245 255 250
Azure	F0 FF FF	240 255 255
AliceBlue	F0 F8 FF	240 248 255
GhostWhite	F8 F8 FF	248 248 255
WhiteSmoke	F5 F5 F5	245 245 245
Seashell	FF F5 EE	255 245 238
Beige	F5 F5 DC	245 245 220
OldLace	FD F5 E6	253 245 230
FloralWhite	FF FA F0	255 250 240
Ivory	FF FF F0	255 255 240
AntiqueWhite	FA EB D7	250 235 215
Linen	FA F0 E6	250 240 230
LavenderBlush	FF F0 F5	255 240 245
MistyRose	FF E4 E1	255 228 225

Gray colors

Gainsboro	DC DC DC	220 220 220
LightGrey	D3 D3 D3	211 211 211
Silver	C0 C0 C0	192 192 192
DarkGray	A9 A9 A9	169 169 169
Gray	80 80 80	128 128 128
DimGray	69 69 69	105 105 105
LightSlateGray	77 88 99	119 136 153
SlateGray	70 80 90	112 128 144
Black	00 00 00	0 0 0

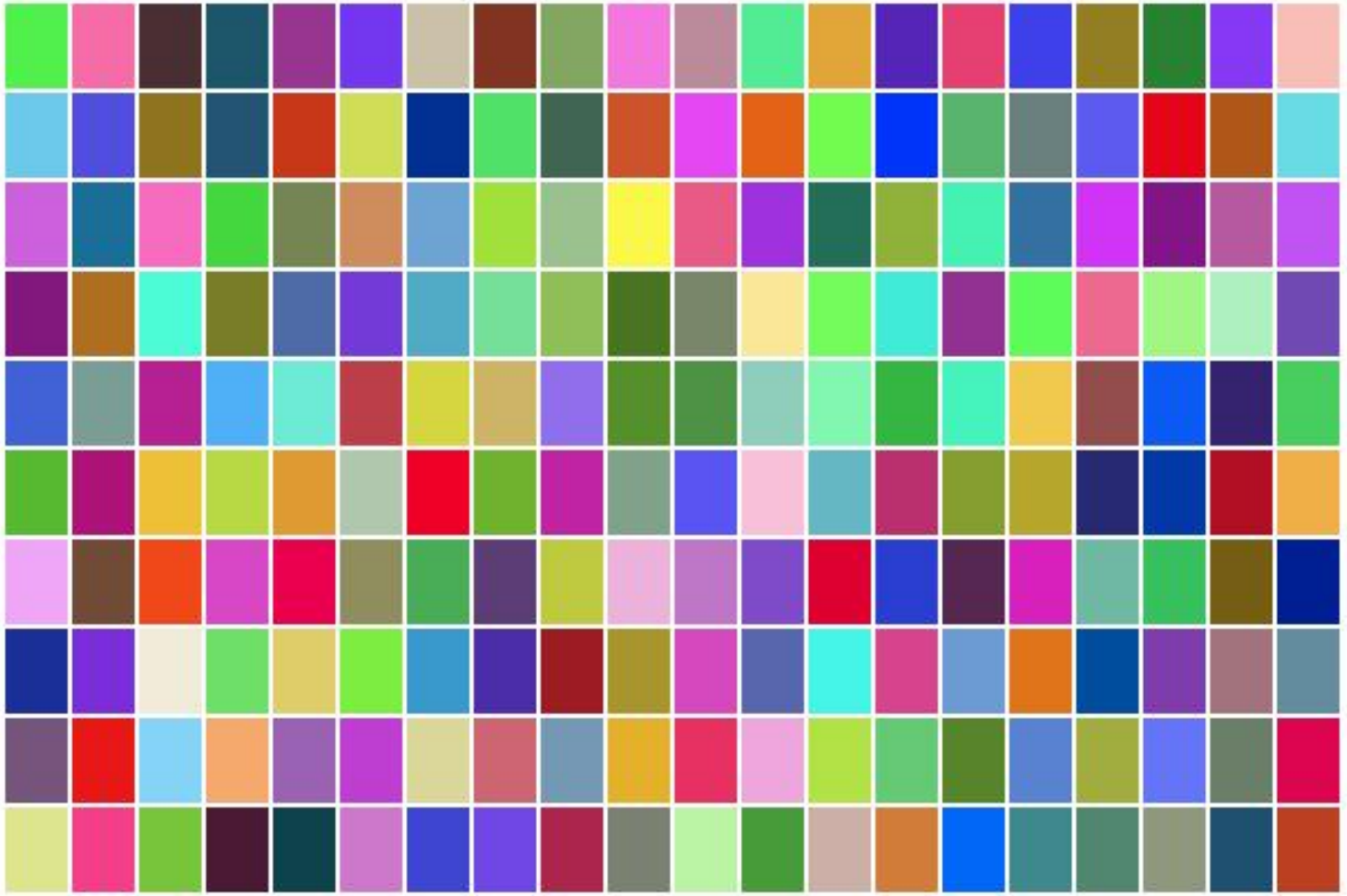
color1.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            HTML canvas fillStyle property
        </title>
    </head>
    <body>
        <canvas id="GFG" width="500" height="300"></canvas
        <script>
            var x = document.getElementById("GFG");
            var contex = x.getContext("2d");
            // set fillStyle color green.
            contex.fillStyle = "green";
            contex.fillRect(50, 50, 350, 200);
            contex.stroke();
        </script>
    </body>
</html>
```


HTML canvas fillStyle property
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Starter/A2%20Elements/color1.html
Apps

color2.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>
```



```
    <body>
       <canvas width="600" height="400" id="canvas" ></canvas>
        <script>
            var canvas = document.getElementById("canvas");
            var ctx = canvas.getContext("2d");
            var CordinateX = 0;
            var CordinateY = 0;
            var Tilemaker = function(ctx,CordX,CordY,maxx,maxy){
                var colorarr = ['1','2','3','4','5','6','7','8','9','0','a','b','c','d','e','f'];
                var w = Math.floor(canvas.width/maxx)-2;
                var h = Math.floor(canvas.height/maxy)-2;
                var CordX0 = CordX;

                for (var i = 0; i < maxy; i++) {
                    for (var j = 0; j < maxx; j++){
                        var Color ='#';
                        for(var k =0; k<6; k++){
                                var r = Math.floor(Math.random()*16);
                                Color = Color+colorarr[r];
                            }
                       ctx.fillStyle = Color;
                       ctx.fillRect(CordX,CordY,w,h);
                       CordX = CordX+w+2;
                    }
                   CordX = CordX0;
                   CordY = CordY+h+2;
                }
            }
            Tilemaker(ctx,CordinateX,CordinateY,20,10);
        </script>
    </body>
```

Gradient

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

Parameters

x0

The x-axis coordinate of the start point.

y0

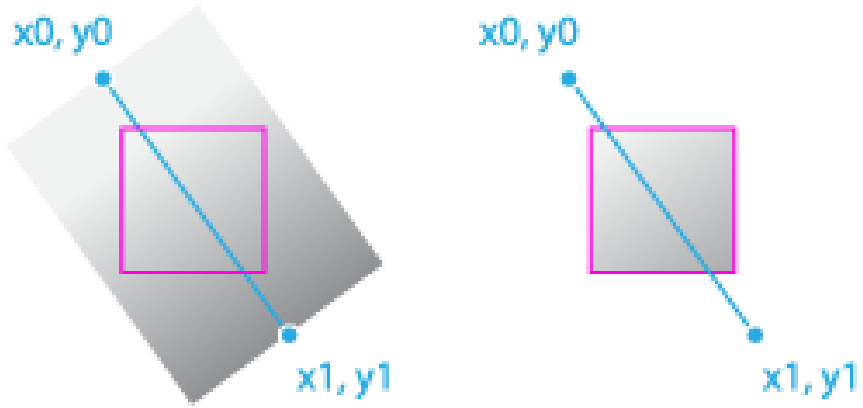
The y-axis coordinate of the start point.

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

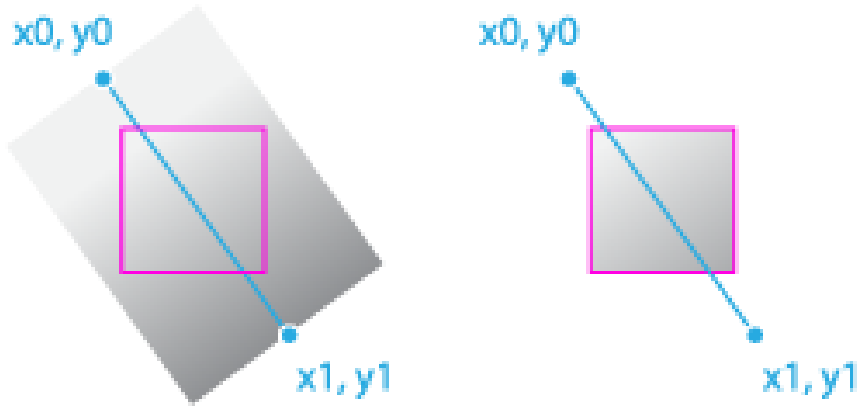
The y-axis coordinate of the start point.

x1

The x-axis coordinate of the end point.

y1

The y-axis coordinate of the end point.



Gradient

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

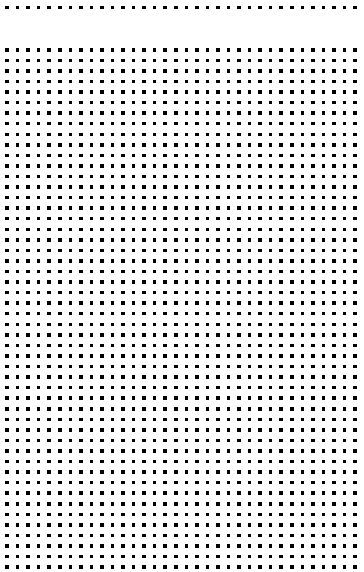
gradient1.html

```
<!DOCTYPE html>
<html>
<head>
    <title>
        HTML canvas fillStyle property
    </title>
</head>
<body>
    <canvas id="GFG"
            width="500"
            height="300">
</canvas>
    <script>
        var x =
            document.getElementById("GFG");
        var contex =
            x.getContext("2d");
        var gr =
            contex.createLinearGradient(50, 0, 350, 0);
        gr.addColorStop(0, "green");
        gr.addColorStop(1, "white");
        contex.fillStyle = gr;
        contex.fillRect(50, 50, 350, 200);
        contex.stroke();
    </script>
</body>
</html>
```

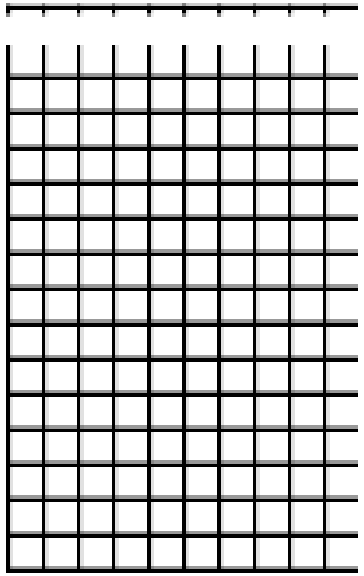
gradient2.html

```html
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.getElementById('canvas');
            var ctx = canvas.getContext('2d');
            // Create a linear gradient
            // The start gradient point is at x=20, y=0
            // The end gradient point is at x=220, y=0
            var gradient = ctx.createLinearGradient(20,0, 220,0);
            gradient.addColorStop(0, 'green');
            gradient.addColorStop(.5, 'cyan');
            gradient.addColorStop(1, 'green');
            ctx.fillStyle = gradient;
            ctx.fillRect(20, 20, 200, 100);
        </script>
    </body>
</html>
```

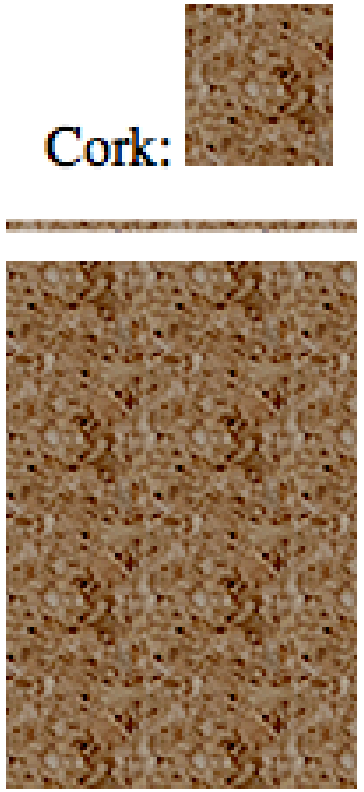
Dot: .



Block: ∟



Cork:



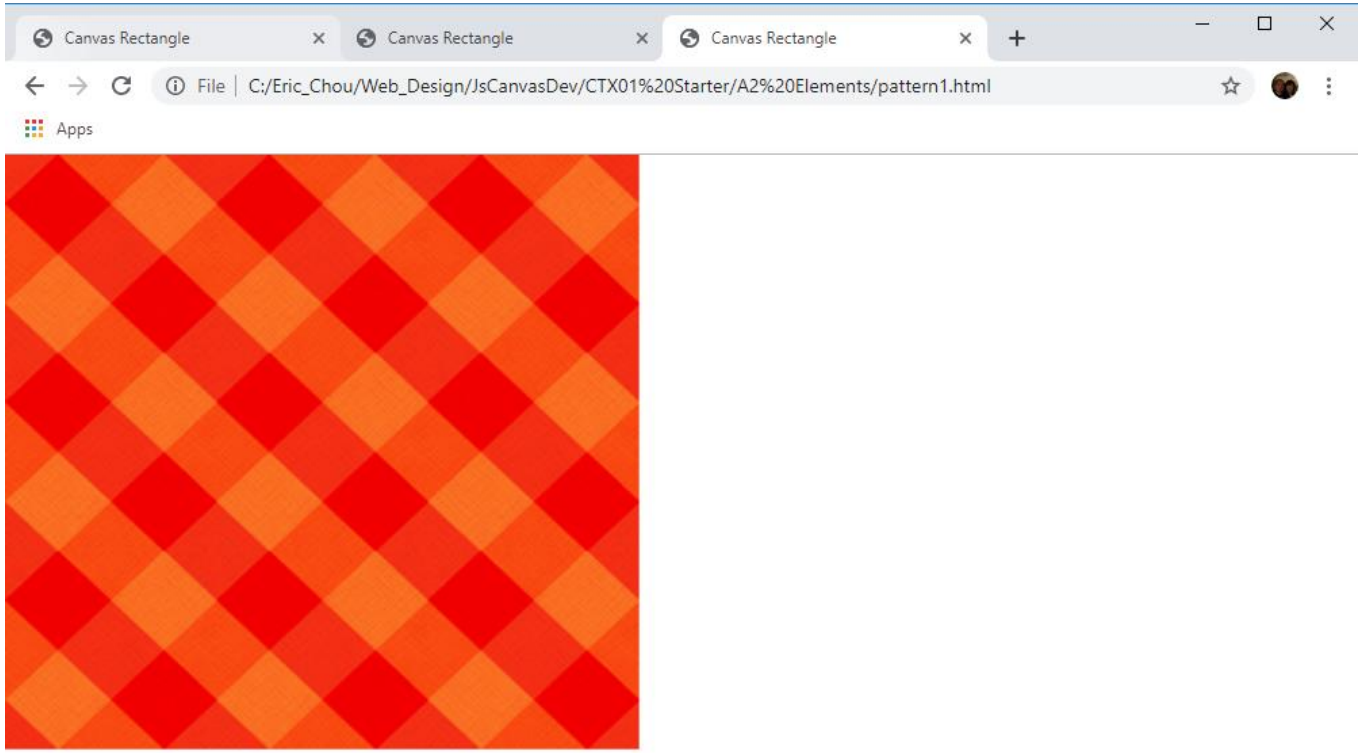
Pattern

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);
}
```

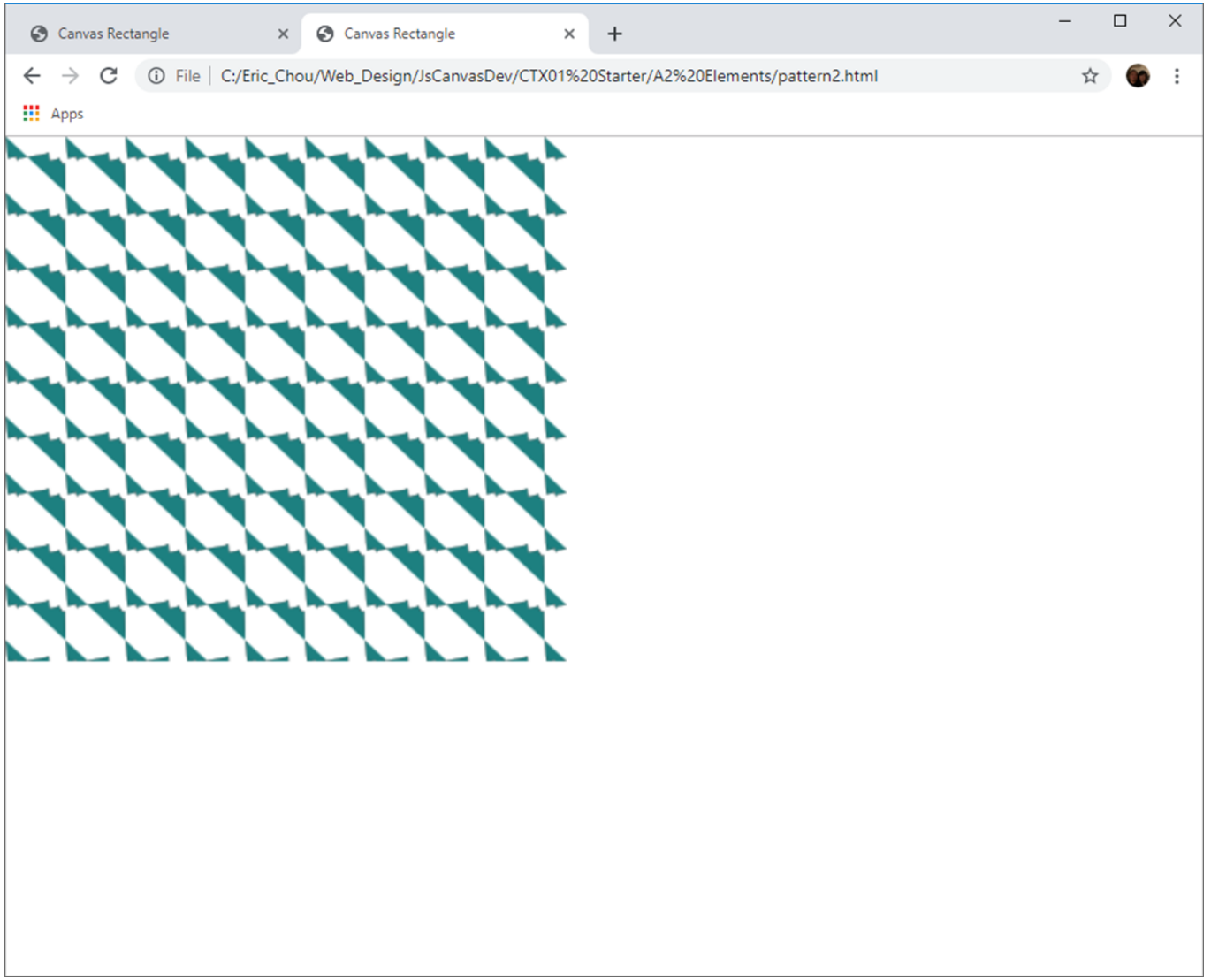
pattern1.html

```
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.getElementById('canvas');
            var ctx = canvas.getContext('2d');
            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);
            }
        </script>
    </body>
</html>
```


Canvas Rectangle
Canvas Rectangle
Canvas Rectangle
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Starter/A2%20Elements/pattern1.html
Apps

pattern2.html

```html
<html>
    <head>
        <title>Canvas Rectangle</title>
        <style>
            body{ margin:0; }
            canvas{
              width:  100%;
              height: 100%;
              background-color:white;
            }
        </style>
        <script type="javascript/text" src="init.js"></script>
    </head>
    <body>
        <canvas id="canvas" width="640px" height="480px"></canvas>
        <script>
            var canvas = document.getElementById('canvas');
            var ctx = canvas.getContext('2d');
            var img = new Image();
            img.src = "bb.png";
            img.onload = function drawPattern() {
                var p = ctx.createPattern(img, "repeat");
                ctx.fillStyle = p;
                ctx.fillRect(0, 0, 300, 300);
            }
        </script>
    </body>
</html>
```


Canvas Rectangle
Canvas Rectangle
File | C:/Eric_Chou/Web_Design/JsCanvasDev/CTX01%20Starter/A2%20Elements/pattern2.html
Apps

STYLES & COLORS

COLORS:

fillStyle = color
strokeStyle = color

Transparency:

globalAlpha = transparencyValue

Line styles:

lineWidth = value
lineCap = type

lineJoin = type

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